SLTS Kernel and Base-Layer Development in the Civil Infrastructure Platform

Yoshitake Kobayashi
Embedded Linux Conference, Portland, February 21-23, 2017
Our Civilization is Run by Linux

https://www.airpano.com/360Degree-VirtualTour.php?3D=San-Francisco-USA
Transport
- Rail automation
- Automatic ticket gates
- Vehicle control

Energy
- Power Generation
- Turbine Control

Industry
- Industry automation
- Industrial communication
- CNC control

Others
- Healthcare
- Building automation
- Broadcasting

https://www.airpano.com/360Degree-VirtualTour.php?3D=San-Francisco-UsA
But there are issues to be solved…

https://www.airpano.com/360Degree-VirtualTour.php?3D=San-Francisco-USA
A Railway System:
25-50 years products life-cycle
with very reluctant nature for product update and upgrade of hardware and base software platform

Image: http://www.deutschebahn.com/contentblob/10862328/20160301+Stw+M%C3%BClheim+Innenansicht+1+(1)/data.jpg
<table>
<thead>
<tr>
<th>Time Period</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 – 5 years</td>
<td>Development time</td>
</tr>
<tr>
<td>2 – 4 years</td>
<td>Customer specific extensions</td>
</tr>
<tr>
<td>1 year</td>
<td>Initial safety certifications / authorization</td>
</tr>
<tr>
<td>3 – 6 months</td>
<td>Safety certifications / authorization for follow-up releases</td>
</tr>
<tr>
<td>(depending on amount of changes)</td>
<td></td>
</tr>
<tr>
<td>25 – 50 years</td>
<td>Lifetime</td>
</tr>
</tbody>
</table>
What we have done on Linux for civil infrastructure systems

• Improve real-time performance and test
• Improve reliability and test
• Improve security and test
• Improve stability and test
• Create a lot of documents and review
  • Open source software licenses compliance
  • Export control classification

• Then, support for long-time such as 20-60 years
• …
We have a problem…
The Problems we face …

• The systems that support our modern civilization need to **survive for a VERY LONG TIME**. Until now the corresponding industrial grade super long term maintenance has been **done by each individual companies**.

• These systems not only have to survive for a long time, they must be **“INDUSTRIAL GRADE”** (robust, secure and reliable). And at the same time the industry will also need to **catch up with the latest technology trends**
The Solutions we need ...

• **We need a Collaborative framework** to maintain one same open source based system for many, many, many years to keep it secure, robust and reliable.

• AND most importantly, we need to do this collaboratively in the **upstream communities**, not locally.
CIP is our solution...

Establishing an Open Source Base Layer of industrial-grade software to enable the use and implementation of software building blocks for Civil Infrastructure Systems

https://www.cip-project.org/
Requirements for the Civil infrastructure systems

Industrial Grade
- Reliability
- Functional Safety
- Security
- Real-time capabilities

Sustainability
- Product life-cycles of 10 – 60 years

Conservative Upgrade/Update Strategy
- Firmware updates only if industrial grade is jeopardized
- Minimize risk of regression
- Keeping regression test and certification efforts low

This has to be achieve with ...

Maintenance costs
- Low maintenance costs for commonly uses software components
- Low commissioning and update costs

Development costs
- Don’t re-invent the wheel

Development time
- Shorter development times for more complex systems
Things to be done: Creation of “Open Source Base Layer”

Open Source Base Layer

- Open source based reference implementation
- Strat from a minimal set for the controllers in the industrial grade systems

Non-CIP packages
Any Linux distribution (e.g. Yocto Project, Debian, openSUSE, etc.) may extend/include CIP packages.

CIP Reference Filesystem image with SDK (CIP Core packages)

CIP SLTS Kernel

CIP Reference Hardware
Scope of activities

User space

- Kernel space
  - Linux Kernel
  - Middleware/Libraries
    - Safe & Secure Update
    - Monitoring
    - Security
    - Real-time support
    - Real-time / safe virtualization
  - App container infrastructure (mid-term)
  - App Framework (optionally, mid-term)
    - Domain Specific communication (e.g. OPC UA)
    - Shared config. & logging
    - Multimedia

Middleware/Libraries

Tools
- Build environment (e.g. yocto recipes)
- Test automation
- Tracing & reporting tools
- Configuration management
- Device management (update, download)
- Application life-cycle management

Concepts
- Functional safety architecture/strategy, including compliance w/ standards (e.g., NERC CIP, IEC61508)
- Long-term support Strategy: security patch management
- Standardization collaborative effort with others
- License clearing
- Export Control Classification

On device software stack

Product development and maintenance

- App container infrastructure (mid-term)
- App Framework (optionally, mid-term)
- Domain Specific communication (e.g. OPC UA)
- Shared config. & logging
- Multimedia
- Safe & Secure Update
- Monitoring
- Security
- Real-time support
- Real-time / safe virtualization

15
Technical topics and related projects (Feb. 2017 version)

* Topics will be added or removed to reflect CIP technical interests

**Application support**
- App Framework
- HMI Framework
- FW update
- App deployment

**Middleware / Tools**
- Coherent Security Mechanisms
  - Configuration/Device management
    - Configuration
    - Industrial Zeroconf
  - Multimedia
  - Common issues
    - Y2038

**Linux Kernel**
- Security
  - Anomaly detection
  - LSM
  - SELinux
- Functional Safety
  - Monitoring/error detection
    - SIL2LinuxMP
    - SIL3 support
  - Live patching
- Userland Isolation
  - LXC
  - Cgroups
- Kernel Isolation
  - SafeG
  - Jailhouse
  - Communication
- Real-time support
  - GPGPU/FPGA real-time
  - Xénomai
- Monitoring / Tracing
  - Ftrace
  - ktap
  - PREEMPT-RT
  - RAS
- RTOS
  - PREEMPT-RT

**Infrastructure and Services**
- Build and production
  - Yocto Project
  - Debian build system
- Testing
  - LTP
  - kselftest
  - KernelCI
  - Fuego
- Support
  - SLTS
  - Backwards compatibility
- Development process
  - SIL2 support
  - SIL3 support
- Legal topics
  - SPDX
  - FOSSology
- License Clearing
  - Export Control

**Hardware / SoC (x86 or ARM based)**

**Legend**
- To be specified / implemented by CIP
- Integration / cooperation
Scope of activities

User space
- App container infrastructure (mid-term)
- App Framework (optionally, mid-term)

Middleware/Libraries
- Domain Specific communication (e.g. OPC UA)
- Shared config. & logging
- Multimedia
- Safe & Secure Update
- Monitoring
- Security
- Real-time support
- Real-time / safe virtualization

Kernel space
- Linux Kernel

Tools
- Build environment (e.g. yocto recipes)
- Test automation
- Tracing & reporting tools
- Configuration management
- Device management (update, download)
- Application life-cycle management

Concepts
- Functional safety architecture/strategy, including compliance w/ standards (e.g., NERC CIP, IEC61508)
- Long-term support Strategy: security patch management
- Standardization collaborative effort with others
- License clearing
- Export Control Classification

On device software stack
- Product development and maintenance

CIVIL INFRASTRUCTURE PLATFORM

17
Current status of CIP base layer development

• CIP SLTS kernel development
  • Decide the CIP kernel version
    • 4.4 as first CIP kernel. Maintenance expected for 10 years and more (SLTS).
  • Select a maintainer
    • Ben Hutchings as initial CIP-kernel maintainer
  • Define a kernel maintenance policies
    • [https://wiki.linuxfoundation.org/civilinfrastructureplatform/cipkernelmaintenance](https://wiki.linuxfoundation.org/civilinfrastructureplatform/cipkernelmaintenance)
  • Start maintenance
    • Linux 4.4.48-cip2 released on 10th February 2017
  • Create CIP kernel test framework

• CIP core package development
  • Define an initial component set
  • Define component version
  • Contribute to upstream project
  • Start maintenance for SLTS
CIP SLTS Kernel Development
Overview of CIP SLTS kernel

• Kernel trees
  • CIP SLTS (linux-4.4.y-cip)
    • Official CIP SLTS kernel tree
      • https://git.kernel.org/cgit/linux/kernel/git/bwh/linux-cip.git/
      • Based on linux-stable.git
    • Maintainer: Ben Hutchings
    • Validation will be done by CIP
  • CIP SLTS+PREEMPT_RT (will be separately maintained by CIP members)
    • CIP kernel tree based on linux-stable-rt and patches from CIP SLTS
    • Validation will be done by CIP

• Maintenance period
  • 10 years and more (10-20 years)
CIP SLTS Kernel development trees

Mainline

Stable (linux-stable)
- Stable
  - CIP SLTS (linux-4.4.y-cip)
  - Stable-rt
    - CIP SLTS-rt/with FB

Backported patches
- Maintained by Ben Hutchings

1. Feature backports
- Take over from maintainer
- Follow the CIP SLTS with PREEMPT_RT

2. Security fix only
- Take over from maintainer

+PREEMPT_RT
- Validate by CIP members
CIP SLTS Kernel development

• Kernel maintenance policy
  • [https://wiki.linuxfoundation.org/civilinfrastructureplatform/cipkernelmaintenance](https://wiki.linuxfoundation.org/civilinfrastructureplatform/cipkernelmaintenance)
  • Follow the stable kernel development rule as the basis
  • Feature backports are acceptable
    • All features has to be in upstream kernel before backport to CIP kernel
    • **CIP has “Upstream first” policy**
    • Validation will be done by CIP test infrastructure and/or members

• Current backported features on 4.4.y-CIP
  • Kernel Self Protection Project related features
    • Address Space Layout Randomization for user space process (ASLR)
    • GCC’s undefined behaviour Sanitizer (UBSAN)
    • Faster page poisoning
Out-of-tree drivers

• In general, all out-of-tree drivers are unsupported by CIP
• Users can use CIP kernel with out-of-tree drivers
  • If a bug is found in such a modified kernel, users will first demonstrate that it exists in the CIP kernel source release in order for the CIP maintainers to act on it.
Major version release cycle (Next CIP SLTS kernel version)

• CIP will take a LTS kernel every 2-4 years
• Planning to synchronize with LTSI for next CIP SLTS kernel
  • LTSI: http://ltsi.linuxfoundation.org/
CIP testing
Purpose of CIP testing

- Detecting bugs
- Detecting regressions
- Provide test results in a timely manner
Milestones of CIP testing and current status

1. **Board at desk - single dev**
   - A setup that allows a developer to test the CIP kernel on the CIP selected hardware platform connected locally to her development machine using kernelCI tools.

2. **CIP kernel testing**
   - Test the CIP kernel on a regular basis and share the results with other CIP community members.

3. **Define kernel testing as a service within CIP**
   - Define the testing environment within CIP assuming that, in some cases, some members may share the tests, test results or laboratories while others may not.

4. **From kernel testing to system testing**
   - Once the testing environment has been ready and works for the kernel, explore how to extend it to the entire CIP platform.

https://wiki.linuxfoundation.org/civilinfrastructureplatform/ciptesting
CIP kernel testing: Board at desk - single dev

• Goal
  • Create and publish a VM image that contains KernelCI & LAVA
  • Single developer can test the CIP kernel (or any other kernels)

• Current status
  • Kernel CI and LAVA have been merged into one VM
  • Beta version just released!
  • https://gitlab.com/cip-project/board-at-desk-single-dev

• Next step
  • Collaborate with other testing projects such as kernelCI, LAVA and Fuego
  • CIP members plans to join Fuego BoF @ ELC (Thursday 12:10pm at Skyline II)
CIP Core package Development
Current status of Base layer development

1. Define an initial component set
2. Define component version
3. Contribute to upstream project
4. Start maintenance for SLTS
Current status of Base layer development

1. Define an initial component set
   1.5 Talk to upstream maintainer
2. Define component version
3. Contribute to upstream project
4. Start maintenance for SLTS
Initial component set for CIP base layer

CIP Start from a minimal set of packages. “CIP kernel” and “CIP core” packages run on hardware.

Candidates for initial component set

- **Kernel**
  - Linux kernel 4.4 + backported patches
  - PREEMPT_RT patch

- **Bootloader**
  - U-boot

- **Shells / Utilities**
  - Busybox

- **Base libraries**
  - Glibc

- **Tool Chain**
  - Binutils
  - GCC

- **Security**
  - OpenSSL

Keep these packages for Reproducible build

- **Dev packages**
  - Flex
  - Bison
  - autoconf
  - automake
  - bc
  - bison
  - Bzip2
  - Curl
  - Db
  - Dbus
  - Expat
  - Flex
  - gawk
  - Gdb

- **Kernel**
  - Git
  - Glib
  - Gmp
  - Gzip
  - gettext
  - Kbd
  - Libibverbs
  - Libtool
  - Libxml2
  - Mpclib
  - Mpfr4
  - Ncurses
  - Make
  - M4

- **pax-utils**
- **Pciutils**
- **Perl**
- **pkg-config**
- **Popt**
- **Procps**
- **Quilt**
- **Readline**
- **sysfsutils**
- **Tar**
- **Unifdef**
- **Zlib**

NOTE: The maintenance effort varies considerably for different packages.
CIP Project X (Project name is tentative)

• Started an incubation project for minimum base system
  • This project will provide the way to test the installable image

• Goal
  • Input: Debian sources and cip kernel
  • Build mechanism: bitbake and/or Debian build system
  • Output: Minimum deployable base system
CIP will increase the development effort to create an industrial grade common base-layer.

### Development plan

**Phase 1:**
- Define supported kernel subsystems, arch.
- Initial SLTS component selection
- Select SLTS versions
- Set-up maintenance infrastructure (build, test)

**Phase 2:**
- Patch collection, stabilization, backport of patches for CIP kernel packages
- Support more subsystems
- Additional core packages

**Phase 3:**
- Domain specific enhancements, e.g. communication protocols, industrial IoT middleware
- Optionally: more subsystems
- Optionally: more core packages
Summary

• Selected the first CIP kernel and initial maintainer
  • 4.4 as first CIP kernel. Maintenance expected for above 10 years (SLTS).
  • Ben Hutchings as initial CIP kernel maintainer.
  • Define CIP Kernel maintenance policies.

• Defined initial board platforms and provide support for them.
  • Beaglebone Black and (RENESAS BOARD) as initial boards.

• CIP kernel testing
  • Board @ desk - single developer.
  • Kernel CI and LAVA have been merged into one VM.

• Started CIP Project X
  • Goal: create a minimum deployable base system.
Next steps
Next step by CIP

• Board @desk - Single dev
  • Release kernelci VM and test CIP kernel in the open within CIP group.
  • Increase test coverage.
  • Define milestone 2.

• Improve integration with Fuego and LAVA.

• Kernel maintenance: define next steps.

• Analysis: select additional software as part of CIP base layer.

• Collaboration: kernelci.org, Fuego, y2038, KSPP, Real-Time Linux
Please Join us!
Why join CIP?

• **Steer**
  participate in project decisions and technical direction.

• **Participate**
  bring your use cases and ideas to the right forum.

• **Learn**
  by working on daily basis in the open with others with common interest.

• **Collaborate**
  share effort and knowledge. Stand on the shoulders of giants.
Contact Information and Resources

To get the latest information, please contact:

- Noriaki Fukuyasu: fukuyasu@linuxfoundation.org

Other resources

- CIP Web site: https://www.cip-project.org
- CIP Mailing list: cip-dev@lists.cip-project.org
- CIP Wiki: https://wiki.linuxfoundation.org/civilinfrastructureplatform/
- Collaboration at CIP: http://www.gitlab.com/cip-project
Call for new participants!

Provide a super long-term maintained industrial-grade embedded Linux platform.

Platinum Members

Silver Members
Questions?
Thank you!