Time is ready for the Civil Infrastructure Platform

Yoshitake Kobayashi, Toshiba
Urs Gleim, Siemens AG
Embedded Linux Conference Europe, Berlin, October 13, 2016
Definition

Civil Infrastructure Systems are technical systems responsible for supervision, control, and management of infrastructure supporting human activities, including, for example,

- Electric power generation
- Energy distribution
- Oil and gas
- Water and wastewater
- Healthcare
- Communications
- Transportation
- Collections of buildings that make up urban & rural communities.

These networks deliver essential services, provide shelter, and support social interactions and economic development. They are society's lifelines.1)

1) adapted from [https://www.ce.udel.edu/current/graduate_program/civil.html](https://www.ce.udel.edu/current/graduate_program/civil.html)
Linux is widely used in ...
## Civil infrastructure systems

<table>
<thead>
<tr>
<th>Core characteristics</th>
<th>Business needs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Industrial grade</strong></td>
<td><strong>Maintenance costs</strong></td>
</tr>
<tr>
<td>▪ Reliability</td>
<td>▪ Low maintenance costs for commonly</td>
</tr>
<tr>
<td>▪ Functional Safety</td>
<td>uses software components</td>
</tr>
<tr>
<td>▪ Security</td>
<td>▪ Low commissioning and update costs</td>
</tr>
<tr>
<td>▪ Real-time capabilities</td>
<td><strong>Development costs</strong></td>
</tr>
<tr>
<td><strong>Sustainability</strong></td>
<td>▪ Don’t re-invent the wheel</td>
</tr>
<tr>
<td>▪ Product life-cycles of 10 – 60 years</td>
<td><strong>Development time</strong></td>
</tr>
<tr>
<td><strong>Conservative update strategy</strong></td>
<td>▪ Shorter development times for more complex systems</td>
</tr>
<tr>
<td>▪ Firmware updates only if industrial grade is jeopardized</td>
<td><strong>Development time</strong></td>
</tr>
<tr>
<td>▪ Minimize risk of regression</td>
<td>▪ Shorter development times for more complex systems</td>
</tr>
<tr>
<td>▪ Keeping regression test and certification efforts low</td>
<td><strong>Development time</strong></td>
</tr>
</tbody>
</table>

**Conservative update strategy**
- Firmware updates only if industrial grade is jeopardized
- Minimize risk of regression
- Keeping regression test and certification efforts low
The evolution of civil infrastructure systems

Proprietary nature
- Systems are built from the ground up for each product
- Little re-use of existing software building blocks
- Closed systems

Commoditization
- Increased utilization of commodity (open source) components, e.g., operating system, virtualization
- Extensibility, e.g., for analytics

Stand-alone systems
- Limited vulnerability
- Updates can only applied with physical access to the systems
- High commissioning efforts

Connected systems
- Interoperability due to advances in machine-to-machine connectivity
- Standardization of communication
- Plug and play based system designs
Things to be done

• Join forces for commodity components
  • Ensure industrial grade for the operating system platform focusing on reliability, security, real-time capability and functional safety
  • Increase upstream work in order to increase quality and to avoid maintenance of patches

• Share maintenance costs
  • Long-term availability and long-term support are crucial

• Innovate for future technology
  • Support industrial IoT architectures and state-of-the-art machine-to-machine connectivity
Civil infrastructure systems require a super long-term maintained industrial-grade embedded Linux platform for a smart digital future.
Civil Infrastructure Platform aims to provide industrial grade software

Establish an **open source “base layer” of industrial grade software** to enable the use and implementation in infrastructure projects of software building blocks that meet the **safety, reliability, security and maintainability requirements**.

- Fill the gap between capabilities of the existing OSS and industrial requirements.
- Provide reference implementation
- Trigger development of an emerging ecosystem including tools and domain specific extensions

➔ **Initial focus on establishing long term maintenance infrastructure for selected Open Source components, funded by participating membership fees**
Railway Example

3 – 5 years development time

2 – 4 years customer specific extensions

1 year initial safety certifications / authorization

3 – 6 months safety certifications / authorization for follow-up releases (depending on amount of changes)

25 – 50 years lifetime

Image: http://www.deutschebahn.com/content/blob/10862328/20160301+Stw+M%C3%BClheim+Innenansicht+1+(1)/data.jpg
Power Plant Control Example

- 3 – 5 years development time
- 0.5 – 4 years customer specific extensions
- 6 – 8 years supply time
- 15+ years hardware maintenance after latest shipment
- 20 – 60 years product lifetime
Power plant runs on the Linux (please visit our booth)
Why maintaining old kernels?

1. Fear of regressions in newer kernels (performance and system stability)
2. Reducing re-certifications costs and time by minimizing changes
3. Reduced number of kernel versions to be provided by SoC vendors (like LSK or LTSI)
4. Serving as a common base for vendor-specific kernel forks and out-of-tree code (yes, we prefer upstreaming...)

see also http://lwn.net/Articles/700530/
Scope of activities

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

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

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
### Target Systems

<table>
<thead>
<tr>
<th>Networked Node</th>
<th>Embedded Control Unit</th>
<th>Embedded Computer</th>
<th>Embedded Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARM offerings</strong>&lt;sup&gt;1)&lt;/sup&gt;</td>
<td>M0/M0+/M3/M4</td>
<td>M4/7, A9, R4/5/7</td>
<td>ARM A9/A35, R7</td>
</tr>
<tr>
<td><strong>Intel offerings</strong>&lt;sup&gt;1)&lt;/sup&gt;</td>
<td>Quark MCU</td>
<td>Quark SoC</td>
<td>Atom</td>
</tr>
<tr>
<td>Architecture, clock</td>
<td>8/16/32-bit, &lt; 100 MHz</td>
<td>32-bit, &lt; 1 GHz</td>
<td>32/64-bit, &lt; 2 GHz</td>
</tr>
<tr>
<td>non-volatile storage</td>
<td>n MiB flash</td>
<td>n GiB flash</td>
<td>n GiB flash</td>
</tr>
<tr>
<td>RAM</td>
<td>&lt; 1 MiB</td>
<td>&lt; 1 GiB</td>
<td>&lt; 4 GiB</td>
</tr>
<tr>
<td>HW ref. platform</td>
<td>Arduino class board</td>
<td>Raspberry Pi class board</td>
<td>SoC-FPGA, e.g. Zync</td>
</tr>
<tr>
<td>application examples</td>
<td>Sensor, field device</td>
<td>control systems</td>
<td>special purpose &amp; server based controllers</td>
</tr>
<tr>
<td>PLC</td>
<td></td>
<td></td>
<td>industrial PC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gatesways</td>
<td>multi-purpose controllers</td>
</tr>
</tbody>
</table>

**Out of scope:**
- Enterprise IT and cloud system platforms.

**Reference hardware for common software platform:**
- Start from working the common HW platform (PC)
- Later extend it to small/low power devices

---

1) Typical configurations Q1/2016

Relationship between CIP and other projects

CIP will do not only development for CIP but also fund or contribute to related upstream projects

- Import source code from open source project or existing distribution to CIP
- Backport patches from upstream to CIP version

Open source projects

- Collaborative Projects (e.g. RTL, Yocto, CII)
- Existing project / distro
- Existing project
- New CIP sub-project

CIP Super Long Term Support Project

- Optional: funding of selected projects
- Contribution

Budget

Developers

- CIP developers
- Developers from member companies

Member companies

Existing project / distro

Open source projects (Upstream work)
Upstream first policy for implementation of new features

All deltas to mainline to be treated as technical debt

- Avoid parallel source trees, directly discuss features in upstream projects
- Upstream first for fixes and features, just like for stable kernels
- Afterwards back-port to super long-term versions driven by CIP
Why super long term support? (Let’s look a typical development process)

1. Certification/Authorization (1 year)
2. Development for new system (3-5 years)
3. Customer specific adaptation (2-4 years)
4. BSP Development (3 month – 1 year)
5. Follow-up releases (3-6 month. Depending on changes)
6. Typical LTS support term (2 years)
7. Follow-up releases (3-6 month. Depending on changes)
8. Hardware replacement (?)
9. 20-50 years
**Super Long Term Support - Motivation**

### Upstream Kernel tree
- **Long-term support (LTS)**: Backports bug fixes for 2 years

### CEWG
- **Long-term support Initiative (LTSI)**: Add extra functionality on LTS for embedded systems and support it for 2 years

### Every company, every project
- Backport of bug fixes and hardware support: the same work is done multiple times for different versions.

- **Approx. 2-5 years**
- **About 3 months**
- **10 years – 15 years**

**Release / Maintenance release**
### CIP kernel super long term support (SLTS) overview

<table>
<thead>
<tr>
<th>Kernel.org</th>
<th>CEWG</th>
<th>CIP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Upstream</strong></td>
<td><strong>Long-term support (LTS)</strong></td>
<td><strong>CIP super long-term supported kernel</strong></td>
</tr>
<tr>
<td><strong>Kernel tree</strong></td>
<td><strong>Backports bug fixes for 2 years</strong></td>
<td><strong>Need to be maintained more than 10 years</strong></td>
</tr>
<tr>
<td>Approx. 3 months</td>
<td>Approx. 2-5 years</td>
<td>Goal: 10 years - 15 years</td>
</tr>
<tr>
<td>Approx. every 3 years</td>
<td>Approx. every 3 years</td>
<td>Approx. every 3 years</td>
</tr>
</tbody>
</table>

- **Backports**: e.g. for SoC support reviewed by CIP
- **After 5 years merge window for new features will be closed, CIP kernel changes focus to security fixes.**
Announcement: The first CIP SLTS kernel version

4.4
Announcement: The first CIP SLTS kernel version

• CIP will maintain the Linux kernel 4.4 for more than 10 years

• Selection Criteria for the first SLTS kernel version
  • LTS version, ideally synchronized with LTSI
  • Broadly used for civil infrastructure systems
    • Currently deployed products
    • Upcoming products

• Next SLTS kernel version?
  • Will be announced in 2-3 years
  • Synchronize with LTSI kernel version at this timing
Super Long-Term Stable Team

- Ben Hutchings is first super long-term kernel maintainer
  - Well-known Debian contributor and package maintainer
  - Currently LTS maintainer for 3.2 and 3.16
- Ben will be supported by one additional developer
- Work started in September 2016
  - Setup of SLTS development and validation process
  - Prepare and perform first SLTS kernel release
  - Support CIP in extending SLTS model to further core packages
Plans for CIP SLTS kernel development

• Development Process
  • Development process will be similar to LTSI
    • Accept feature backports from upstream kernel
    • CIP will have merge windows and validation periods for feature backporting
  • Important NOTE: If the backport changes the kernel API, it will not be accepted

• Validation
  • Establishing kernel test infrastructure
  • Enhance on-target testing beyond boot-tests
  • Share the results for open spec boards
CIP Testing Considerations

Testing goals

• Perform testing on real HW (VM: no detail quirks and real-world issues)
• Focus on CIP reference platforms
• Critical Fixes: Build & test within hours on all machines
• No continuous functional testing (for instance, latencies)
• Super-Long-Term result preservation
• Align approach with established community best practices
CIP Testing Considerations (cont’d)

Current Status

• Initial CIP-private instance of Kernel CI (vagrant based)
  • Member companies can run local labs
  • HW rack standard (standardized physical and electrical setup) under consideration
• Purely local operation; results via central public web server once fully operational
• Job + Build scheduling: To be defined (likely Fuego and friends)
• Feed results back to Kernel CI?

Kernel-CI: https://kernelci.org/
Fuego: http://elinux.org/Fuego
Selection Criteria for Userspace Packages

• Essential for booting and basic functionality
• Commonly used in civil infrastructure systems
• Security sensitive
• Likely maintainable over 10 years+ period
• We are open for proposals!
## Further Candidates for Super Long-term Maintenance

### An Example minimal set of “CIP kernel” and “CIP core” packages for initial scope

<table>
<thead>
<tr>
<th>Super Long-term support</th>
<th>Maintain for Reproducible build</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kernel (SLTS)</strong></td>
<td></td>
</tr>
<tr>
<td>- Kernel</td>
<td>- Flex</td>
</tr>
<tr>
<td>- Linux kernel (cooperation with LTSI)</td>
<td>- Git</td>
</tr>
<tr>
<td>- PREEMPT_RT patch</td>
<td>- Glib</td>
</tr>
<tr>
<td>- Bootloader</td>
<td>- Bison</td>
</tr>
<tr>
<td>- U-boot</td>
<td>- autoconf</td>
</tr>
<tr>
<td>- Shells / Utilities</td>
<td>- automake</td>
</tr>
<tr>
<td>- Busybox</td>
<td>- bc</td>
</tr>
<tr>
<td>- Base libraries</td>
<td>- bison</td>
</tr>
<tr>
<td>- Glibc</td>
<td>- Bzip2</td>
</tr>
<tr>
<td>- Tool Chain</td>
<td>- Curl</td>
</tr>
<tr>
<td>- Binutils</td>
<td>- Db</td>
</tr>
<tr>
<td>- GCC</td>
<td>- Dbus</td>
</tr>
<tr>
<td>- Security</td>
<td>- Expat</td>
</tr>
<tr>
<td>- Openssl</td>
<td>- Flex</td>
</tr>
<tr>
<td>- Openssh</td>
<td>- gawk</td>
</tr>
<tr>
<td></td>
<td>- Gdb</td>
</tr>
<tr>
<td></td>
<td>- M4</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Core Packages (SLTS)</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dev packages</strong></td>
<td><strong>Maintain for Reproducible build</strong></td>
</tr>
<tr>
<td></td>
<td>- pax-utils</td>
</tr>
<tr>
<td></td>
<td>- Pciutils</td>
</tr>
<tr>
<td></td>
<td>- Perl</td>
</tr>
<tr>
<td></td>
<td>- pkg-config</td>
</tr>
<tr>
<td></td>
<td>- Procps</td>
</tr>
<tr>
<td></td>
<td>- Quilt</td>
</tr>
<tr>
<td></td>
<td>- Readline</td>
</tr>
<tr>
<td></td>
<td>- sysfsutils</td>
</tr>
<tr>
<td></td>
<td>- Tar</td>
</tr>
<tr>
<td></td>
<td>- Unifdef</td>
</tr>
<tr>
<td></td>
<td>- Zlib</td>
</tr>
</tbody>
</table>

**NOTE:** The maintenance effort varies considerably for different packages.
Development plan

CIP will increase the development effort to create industrial grade common base-layer

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
Currently under discussion

• CIP should collaborate with other similar efforts
  • LSK (Linaro Stable Kernel)
  • Other distributor or SoC vendors

• Selection of features for backporting
  • PREEMPT_RT
    • PREEMPT_RT might be merged into separate branch
  • KSPP (Kernel Self Protection Project)

• Testing infrastructure (KernelCI + Fuego)

• Kernel maintenance policy

• Userland package selection
Milestones

• 2016:
  • Project launched announcement at Embedded Linux Conference 2016
  • Requirements defined, base use cases defined, technical & non-technical processes established (license clearing, long-term support), maintenance plan
  • Common software stack defined, related core projects agreed (e.g. PREEMT_RT, Xenomai), maintenance infrastructure set up
  • Domain specific extensions defined, tool chain defined, test strategy defined
  • Maintenance to be operational and running

• 2017:
  • Realization phase of selected components

• 2018:
  • Advancement, improvements, new features
Please join!

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

Current members

Platinum Members

Silver Members

HITACHI  
SIEMENS  
TOSHIBA
Why join CIP?

• Participate in **project decisions** through the governing board and/or committees; leverage an ecosystem of like-minded participants to help drive project priorities as a community.

• Provide **technical direction** through a TSC representative enabling fast engagement and input into the technical direction of the project.

• Demonstrate support for CIP.

• Priority access to any events, sponsorship and marketing opportunities. Potential events include:
  • Embedded Linux Conference
  • LinuxCon
  • Collaboration summits
  • Other community events

• Visibility on the CIP website and in membership collateral
Contact Information and Resources

To get the latest information, please contact:

- Noriaki Fukuyasu  
  fukuyasu@linuxfoundation.org
- Urs Gleim  
  urs.gleim@siemens.com
- Yoshitake Kobayashi  
  yoshitake.kobayashi@toshiba.co.jp
- Hiroshi Mine  
  hiroshi.mine.vd@hitachi.com

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/
Questions?
Thank you!