LINUX SAFETY VERIFICATION
A PROCESS FOR USING LINUX IN SAFETY-CRITICAL ENVIRONMENTS
MOTIVATION
Main message: Software innovations will disrupt every industry, even very established industries.

Various companies in the mechatronics industry are struggling with this change being driven by software innovations and software industry competitors.

Give profits to the software vendors or invest to explore alternatives?
THE HISTORY OF UNIX

1980s
Various UNIX operating systems in the market. Vendors largely controlling their users, with an incompatible vendor-lock-in mess.

1990s
Linux is born. Linux forms as the coalition of those tired users and losers. With an open-source collaboration model, Linux builds a strong ecosystem of users and software companies.

Today
Linux (in 2017):
- 90% public cloud workload
- 62% embedded market share
- 99% supercomputer market share
- 82% world’s smartphones

(Source: https://www.linuxfoundation.org/publications/2017-state-of-linux-kernel-development/)

Facebook, Google, IBM, Intel... have Linux kernel teams for their Linux operating system. They are in control of the software chain & stack, although they don’t even sell software, but services or hardware.
HISTORY REPEATS ITSELF

Mechatronic industry is now at the same crossway for safety-critical operating systems to run complex algorithms and software.

How to create a healthy ecosystem of safety-critical operating systems to focus on innovative software functions?
The Linux kernel has:
- Large Development Ecosystem
- Security Capabilities
- Multi-Core Support
- Unmatched Hardware Support
- Many Linux Experts at all levels available

The Linux kernel is missing:
- Real-time Capabilities
- Proven Safety-compliant Development Process

Can these gaps be closed?
THE OSADL SIL2LINUXMP PROJECT

– **Mission:**
  – **Provide procedures and methods** to qualify Linux on a multi-core embedded platform at safety integrity level 2 (SIL2) according to IEC 61508 Ed 2.
  – **Show feasibility** of **procedures and methods** on a real-world system
  – **Show potential** for collaboration and re-use of Linux kernel analysis

– **Collaborative project of industrial & research partners**
  – project running since 2015, organized by OSADL
  – Full members: BMW Car IT, Intel (since ‘17), A&R Tech, KUKA, Sensor-Technik Wiedemann (full members till ’16, reviewing members in ‘17)
  – Reviewing members: Bosch, Elektrobit, Hitachi, Linutronix, MBDA Italia, MEN Mikro Elektronik, Mentor, OpenSynergy, Pilz GmbH & Co. KG, Renesas, Vienna Water Monitoring Solutions
  – Academic members: Alexey Khoroshilov (ISP RAS), Kinggo Chow (Lanzhou Univ.), Julia Lawall (Inria/LIP6), Frank Tränkle (HS Heilbronn)
  – Experts from certification bodies: Bernhard Nölte (TÜV Süd), Oliver Busa, Robert Heinen, Hendrik Schäbe (TÜV Rheinland)
  – SIL2LinuxMP core working team: Nicholas McGuire, Andreas Platschek, Lucas Böhm, Markus Kreidl (OpenTech)
GENERAL ASSUMPTIONS
Using Linux in a Safety-Critical System

Unsafe

Safe

Objectively Safe

according to global safety standards

Establishedly Safe

following a common unquestioned interpretation of safety standards
First Law of System Safety Engineering

A System is safe by a proper composition of system elements.

Corollary

A System with Linux is safe if Linux is composed properly into the overall system.

Task: Find out how to compose Linux properly into a system
   – Determine which safety capabilities can be argued for Linux?
   – Find proper balance between assigned safety requirements and feasible capabilities
   – Provide full evidences for required capabilities
POSSIBLE SYSTEM ARCHITECTURES

**System Architecture A**

- High-performance Software
- Safety Check SW
- COTS hardware
- High-integrity HW

Computes output: input → output
Checks output: output → checked output

Assumptions/Drawbacks:
- Needs decomposition of safety of complex function to a simple checking on lower performance high-integrity HW
- Checking must detect subtle errors from HW and OS of high-performance computation.

**System Architecture B**

- Non safety-critical Software
- High-performant, Safety Software
- Customer market OS
- Hypervisor
- COTS hardware

Assumptions/Drawbacks:
- Hypervisor & HW guarantees isolation
- Safety SW without underlying OS
- If needed, scheduling, multi-threading & file system is implemented in safety SW

**System Architecture C**

- Non safety-critical Software
- High-performant, Safety Software
- Hypervisor
- COTS hardware
- Linux kernel

Assumptions/Drawbacks:
- Linux kernel provides sufficient isolation
- Safety SW uses Linux scheduling, multi-threading, file system etc.
- Parts of Linux kernel and HW functionality must be qualified.

SIL2LinuxMP project focuses on work for system architecture C.
NOTABLE FACTS ON THE LINUX KERNEL

- Linux is a large software project with an impressive change rate
  - 23 million lines of code & 2 million lines of documentation
  - 14,000 commits in every release/every 70 days
  - 17,000 developers in total history, 1,700 developers in each release
  - kernel developers are affiliated with many different companies or act as individuals

- Development process
  - Highly transparent due to public visible collaboration
  - Process defined and enforced by social contract, but not legal working contracts

- Stabilizing phase of Linux LTS kernel versions
  - 7360 bug-fix commits in the 4.9 branch until v4.9.95 => ~100 bugs corrected each week
  - detected by continuous developer review, various verification activities and execution on billion devices
RESEARCH RESULTS
OVERVIEW OF RESULTS

– Qualification Route
– Systematic Approach to Replace Methods
– Hazard-driven Decomposition, Design and Development
– Software Layers-of-Protection Analysis
– SIL2LinuxMP Architecture
– Statistic Modelling
– Verification Activities
– Safety-Impact Analysis
– **Goal** of the system analysis: **Precise technical safety requirements** on lower levels with traceability on system’s safety impact

– **First naive system safety engineering**
  “The syscall open() is used in a safety-critical application and must work correctly.”
  – Problem: Too imprecise to guide further testing, verification and validation activities

– **Safety engineering with Hazard-driven Decomposition, Design & Development (HD³)**
  – Dedicated method to achieve more precision on complex systems with multiple levels for fault avoidance & detection
  – Results in 12 constraints on syscall open()!
  – Specific testing and verifications under those specific conditions is now feasible

---

**System Call:** open()

**Origin:** I/O Setup (ST.3020-ST.3029), Create File (ST.3040-ST.3049)

**HAZOP IDs:** SD.3520-3D.3529

**References:** 1) man 2 open 2) POSIX 1003.13 summary in section 6.6.1.4 3) open()

**Related SACs:** STS.3013 (minimize access privileges), STS.3014 (access modes pre-determined), STS.3015 (init opens/creates files), STS.3019 (restricted usage of lseek()), STS.3028 (defined file creation flags), STS.3029 (check new files existence), STS.3030 (file creation logged), STS.3034 (file creation in non-RT), SDS.3508 (magic numbers prohibited), SDS.3509 (write files exclusively), SDS.3510 (set and verify permissions), SDS.3512 (file namespace specified), SDS.3513 (random bits in file names)
**HAZARD-DRIVEN DECOMPOSITION, DESIGN & DEVELOPMENT**

**System Safety Engineering Method** to obtain **precise** technical safety requirements on lower levels with **clear safety impact**

**First Naive System Safety Approach**
- Top-level System Design & Hazard Analysis
- Functional Decomposition & System Design over multiple Levels without further Hazard Analysis
- Technical Safety Requirements on Level n

**Hazard-driven Decomposition, Design & Development (HD³)**
- Top-level System Design & Hazard Analysis
- Functional Safety Requirements
- Single-Level System Design
- Hazard Analysis
- Safety Requirements on Level 2
- Single-Level System Design
- Hazard Analysis
- Safety Requirements on Level 3
Independent Protection Layers (IPL) are in Software (mostly security mechanisms).

- IPLs designed to provide isolation, but no quantification possible.

- Independence of Layers assured by analysis of:
  - software architecture and implementation
  - development meta data, e.g., development history and contributors.
Idea:

Confirm process effectiveness qualitatively by statistic modelling of bug defect detection over time.
Still highly preliminary and under discussion:

Needs review of experts in quantative software engineering and statisticians

In other domains, statistics is used intensively to prove facts empirically, but not yet so much in software development processes.

We need to ensure that we do not: „incidentally over-interpret measured data, present data in not so useful ways, and/or draw unsupport conclusions from statistical analyses“

(Prof. Dr. Wolfgang Maurerer, Siemens/OTH Regensburg)
NEEDED QUALITY ASSURANCE ACTIVITIES FOR SAFETY-CRITICAL SYSTEMS

- **Ongoing and future quality assurance activities**
  - **Coding style**
    - Respect existing coding style
    - Provide evidences for its quality
    - Monitor and motivate its compliance
  - **Testing**
    - Extend tests of the Linux Test Project for the determined syscalls
  - **Static analysis**
    - Detect more bug patterns and bug classes with coccinelle, sparse et al.
  - **Change management**
    - Track if bug fixes from main line are consequently backported
    - Analyze which kernel bugs & fixes impact the systems’ safety

- Activities focus on parts of the Linux kernel relevant for the systems’ safety
VERIFICATION TOOLS FOR THE LINUX KERNEL

Syntactic and Static Verification Tools:
- checkpatch.pl
- coccinelle, sparse, and smatch
- Linux Driver Verification (LDV) and Coverity

Dynamic Verification Tools (Testing and Fuzzing):
- kernelci.org and Linux Test Project
- Trinity and syzkaller

- These tools are regularly applied by the kernel community.
- Contributions of a Linux Safety Verification Project:
  - A Linux Safety Verification Project can motivate, support and contribute to activities around those communities.
  - A Linux Safety Verification Project can argue the contribution of such tools against the safety standards (IEC 61508).
MAINTENANCE ACTIVITY: SAFETY IMPACT ANALYSIS

- **Rationale** for doing safety impact cause analysis during operations
  - Safety standard requires continuous monitoring and analysis of identified issues
- **Implementation** with the Linux kernel development
  - Bugs are continuously found in Linux
  - Bug-fix commits are backported to the affected LTS branches
  - Then all product developers must determine if the bug and bug fix impact systems’ safety of each system

- **Process** for each bug-fix:
  - **Step 1** (once for each bug-fix):
    - **Kernel analysis team** describes the impact of a kernel bug on user space and its bug-fix in high detail
    - Analysis independent of the specific system, **one collaborative team**
  - **Step 2** (for each bug-fix and each system):
    - **System analysis team** judges if the described bug-fix impacts the system and the system’s safety.
    - One team for each system employing Linux
CONCLUSION
CURRENT USE CASE STATUS

Process, Methods and Qualification Route

Use Case, System Decomposition, Hazard Analysis

- Hardware Safety Requirements
- Linux Software Safety Requirements
- Hardware Selection
- Kernel Configuration

Final Assessment of System Development

- Integration Activities
- Verification Activities

Kernel Configuration

Software Safety Analysis

✅ Process, Methods, and Qualification Route

Linux Safety Verification | Lukas Bulwahn | June 14th, 2018
NEXT STEPS AND OPEN CHALLENGES

– Work on Safety Development Process:
  – Increase Confidence in Statistic Modelling
    – Consult with experts in quantative software engineering and statisticians
  – Define Maintenance Process

– Find Suitable Hardware with an appropriate Safety Manual
  – e.g., with guarantees for suitable systematic capabilities on the memory management unit (MMU).

– Implement Verification Activities

– Tie Parts together in the project’s Use Case

– Grow Circle of Collaborative Companies to continue investigation and shared maintenance
CONCLUSION

- To learn more, join the SIL2LinuxMP Safety-Critical Linux working group
- Contact Nicholas McGuire at safety-at-osadl.org or contact me at lukas.bulwahn-at-bmw.de

- How you will learn more?
  - Full-day Workshop in October 2018 planned, as satellite event to ELCE 2018 in Edinburgh
  - Various documents of mentioned results are under review of participating companies and will be published soon.

- Using Linux in safety-critical systems is feasible.
  - But still much to learn, understand, interpret and do.

Thanks for your attention!
This presentation is licensed under Creative Commons Attribution 4.0 International License (cf. https://creativecommons.org/licenses/by/4.0/). Here is a human-readable summary of (and not a substitute for) the license.

You are free to:
- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

Under the following terms:
- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

The licensor cannot revoke these freedoms as long as you follow the license terms.

For other uses beyond this license, e.g., under other terms, such as with endorsement of derived work or removal and change of attribution, please contact the author.