Linux in Basic Safety Applications

Jason R. Smith

May 13, 2020

Abstract

The complexity of applications continues to increase, often necessitating software solutions for multitasking, e.g. use of operating systems such as Linux. This white paper considers applications that may now be performing a handful of basic safety-related functions using Linux, and how they can be designed to meet similarly basic safety requirements.

Contents

1 Introduction

1.1 Overview

The complexity of safety-related applications continues to increase, often necessitating software solutions for multitasking, e.g. use of operating systems such as Linux. The embedded electronics and software used in these applications, including operating systems, may be required to comply with functional safety standards such as IEC 61508 and ISO 26262. Example applications include self-driving cars, collaborative robots, etc.
Also worth considering are other applications, also complex in nature and in need of an operating system, that may perform a handful of basic safety-related functions such as voltage, current, and/or temperature monitoring. These applications may not have the same level of risk (severity and/or probability) as the ones previously mentioned, but are safety-related nonetheless. Examples include inverters for renewable energy applications, Battery Management Systems (BMSs), and Electric Vehicle (EV) chargers .
These other applications usually do not require compliance with comprehensive functional safety standards such as IEC 61508, ISO 13849, ISO 26262, etc., but still likely require a level of assurance that any embedded software performing safety-related functions will do so reliably. In these cases, standards such as ANSI/UL 1998, CSA C22.2 No. 0.8, or EN 50271 are used. These three standards have specific requirements pertaining to third-party software including operating systems.
The remainder of this document proposes potential compliance solutions with these standards when Linux is used as an operating system. Each section of this document proposes actions to take for either the Developer, i.e. the software developer of the end product, or ELISA, i.e. the Enabling Linux for Safety Applications project.
While the solutions proposed in this document are not as sophisticated or comprehensive as those that would be needed for IEC 61508, ISO 13849, or ISO 26262, they have a similar objective to the requirements described in those standards. Therefore, some of the proposed solutions could be reused for those standards as well, particularly for lower SILs (SIL 1), PLs (PL a, b, c), or ASILs (ASIL A, B).

1.2 The Standards

ANSI/UL 1998 [ref. 1] is UL's standard for Software in Programmable Components and contains requirements for embedded software performing safety-related functions. It is referenced in over 100 product safety standards and its application is not industry or product specific. ANSI/UL 1998 defines operating systems developed by a third-party as Off-the-Shelf Software or OTS (definition 2.30). Requirements for OTS are primarily contained in ANSI/UL 1998 Clause 13, but other clauses of the standard also indirectly impact OTS.
CSA C22.2 No. 0.8 [ref. 2] is CSA's standard for Safety Functions Incorporating Electronic Technology, which applies to safety-related software. Similar to ANSI/UL 1998, CSA C22.2 No. 0.8 is referenced in many product safety standards and its application is not industry or product specific. CSA C22.2 No. 0.8 also defines operating systems developed by a third-party as Off-the-Shelf Software or OTS. Requirements for OTS are primarily contained in CSA C22.2 No. 0.8 Clause 5.2.8, but other clauses of the standard also indirectly impact OTS.
EN 50271 [ref. 3] is the European Norm containing Requirements and Tests for Apparatus Using Software and/or Digital Technologies used for Electrical Apparatus for the Detection and Measurement of Combustible Gases, Toxic Gases, or Oxygen. Unlike the previous two standards, EN 50271 is specific to one application: detection and measurement of gases. EN 50271 has one clause, 4.3.2, that pertains to re-used or commercial operating systems, including ones that have not been certified to any standard. Additional clauses of the standard also indirectly impact operating systems.

2 Information Requirements

2.1 Developer

Responsible for providing the following information pertaining to the OTS software in a Software Development Plan, Configuration Management Plan, or similar:

- Name of OTS software, its version, and provider [ref. 1, Clauses 13.1a, 13.1b; ref. 2, Clauses 5.2.8a and 7.2.4; ref. 3, Clause 4.3.4]
- Description of purpose and function of OTS software [ref. 1, Clauses 13.1c, 13.1d; ref. 2, Clauses 5.2.8b, 5.6.1.1f, and 7.2.4]

- Interface specification of control and data flows in and out of OTS software [ref. 1, Clause 13.1e; ref. 2, Clauses 5.2.8b, 5.6.1.1f, and 7.2.4]
- References to OTS software documentation for each callable routine used [ref. 1, Clause 13.1f; ref. 2, Clauses 5.2.8b, 5.6.1.1f, and 7.2.4]
- Configuration and build description of OTS software [ref. 1, Clause 12.4]
- Methods and activities to maintain and control changes made to the OTS software, including its configuration [ref. 1, Clauses 12.4 and 14]
- Approach and rationale used to select the OTS software [ref. 3, Clause 4.3.5.3.1j]

2.2 ELISA

Assemble a Manual for each Linux distribution that includes the following:

- Name of the distribution, its version, and provider
- Description of purpose and function of the distribution
- Description of interfaceable elements of the distribution, i.e. API description
- The configuration and build description of the distribution

3 Risk Analysis Requirements

Note: ANSI/UL 1998 and CSA C22.2 No. 0.8 require a Risk Analysis to identify appropriate safety measures, whereas EN 50271 prescribes how risks are addressed by design. While what is required in theory depends on which standards are normative, it is the recommendation of the author that both this section and the Design Requirements section below are considered.

3.1 Developer

Responsible for conducting and documenting a Risk Analysis that determines:

- The set of risks [ref. 1, Clause 3.1a; ref. 2, Clause 5.3.1], based on the safety requirements [ref. 1, Clause 3.2], which considers microelectronic hardware faults [ref. 1, Clauses 8.1 and 8.2; ref. 2, Clause 5.3.2.2], software faults caused by microelectronic hardware faults [ref. 1, Clause 8.3], other states or transitions capable of resulting in a risk [ref. 1, Clause 3.4; ref. 2, Clause 5.3.2.1], and the OTS software [ref. 2, Clause C.4.2.2]

- How the software addresses these risks to acceptable levels [ref. 1, Clause 3.1b; ref. 2, Clauses 5.3.2.3 and 5.3.2.4]

3.2 ELISA

Rationale: a Risk Analysis cannot be conducted at the operating system level without considering the end application. However, failure modes of operating systems such as Linux are known, and it is the author's recommendation to ELISA that foreseeable failure modes of operating systems are communicated clearly to the developer in a Fault Analysis to help facilitate the Developer's Risk Analysis.
Assemble a Fault Analysis that identifies potential failure modes of the operating system, including consideration of:

- Failure of proper allocation of task resources, including scheduling frequency of tasks, criticality of tasks, and resources utilized by the tasks [ref. 1, Clause 6.5, ref. 3, Clause 4.3.2.1a]; this could include, for example:

  - Improper use of pre-emption/prioritization
  - Improper setting of task frequencies/timeout periods
  - Improper memory allocation
  - Improper use of semaphores/mutexes
  - Etc.

- Failure of software partitioning and integrity of partition(s) [ref.1, Clause 7.4]; this could include, for example:

  - Buffer overflows
  - Memory leaks
  - Running out of memory
  - Loss of control of the execution of the software

  - Use of global vs. private variables and functions
  - Etc.

4 Design Requirements

Also see Note above for Risk Analysis Requirements

4.1 Developer

- Must initialize software and product to a risks-addressed or safe state as appropriate for that product [ref. 1, Clauses 7.1 and 9.2]
- All variables in software shall be set to initial values before being used [ref. 1, Clause 6.7]
- A minimum of two instruction sequences (i.e. plausibility checks) shall be employed before transitioning out of a risks-addressed or safe state into an operating state that could be capable of resulting in a risk [ref. 1, Clause 7.7]
- The software shall employ means to identify and respond to states capable of resulting in a risk, such as fail-safe and fault-tolerant concepts, run-time checks, and built-in tests [ref. 1, Clause 6.4; ref. 3, Clause 4.6]; run-time checks and built-in tests shall be conducted automatically as appropriate for the application (EN 50271 Clause 4.6 states once every 24 hours, but this is specific to gas detection applications); more specifically:
  - Program memory shall be protected against single-bit faults and most double-bit faults [ref. 3, Clause 4.6e]
  - Parameter memory and RAM shall be protected against single-bit faults and most double-bit faults, and tested for readability/writability [ref. 3, Clauses 4.6e and 4.6f]
  - Other tests as deemed necessary by the Risk Analysis [ref. 1, Clauses 8.1 and 8.2; ref. 2, Clause 5.3.2.2]
- Means shall be employed to prevent, detect, and resolve non-terminating and non-deterministic states, e.g. division by zero, under/overvoltage condition of power supplies, under/overflow [ref. 1, Clause 6.6]; more specifically:

  - Product shall transition to a risks-addressed or safe state upon power interruption of any duration [ref. 1, Clause 9.1; ref. 3, Clause 4.6a]

- If a fault or any condition capable of resulting in a risk is detected, the product shall transition/remain in a risks-addressed or safe state as appropriate for that product [ref. 1, Clauses 6.1-6.3, 7.6; ref. 3, Clause 4.6]

- Critical and supervisory sections of software shall be partitioned from non-critical sections, or all software is to be treated as critical [ref. 1, Clauses 7.2-7.3]; more specifically:

  - Accessibility of critical instructions and data shall be controlled and protected from being affected by non-critical sections of software [ref. 1, Clauses 7.8-7.9, 7.11]

- Control of the execution of the software shall be maintained and ensured [ref. 1, Clauses 7.5, 9.3, 9.4]; more specifically:

  - Logical and temporal monitoring of the program sequence is provided by monitoring equipment with its own time base [ref. 3, Clauses 4.3.2.2a and 4.6d]
  - The triggering of the monitoring equipment is done through the application software only; not by the operating system [ref. 3, Clause 4.3.2.2b]

- It shall not be possible for the user to modify the configuration of the operating system [ref. 1, Clause 10.2, ref. 3, Clause 4.3.2.1b]

- It shall not be possible for the operating system to automatically update [ref. 3, Clause 4.3.2.1c]; only the manufacturer shall have the ability to do this, fully under their control [ref. 3, 4.3.2.1d]

- Program and parameters shall be stored in non-volatile memory; if program is later executed from volatile memory, it shall be loaded at startup and appropriately checked for validity [ref. 1, Clause 7.10; ref. 3, Clause 4.3.2.1e]

- Safety-related input and output ports are read and controlled by the application software; not by the operating system [ref. 3, Clause 4.3.2.2c and 4.3.2.2d]

4.2 ELISA

In a Design Guide, provide recommendations for safety applications to satisfy the above requirements, including the following in particular:

- Hardware architectural provisions for:
  - Independent voltage supply monitoring
  - Sensors and actuators that are directly monitored and/or controlled by the application software (not the operating system)
  - Independent monitoring device, triggered by the application software (not the operating system), that is capable of maintaining a safe state of the system if the software/OS operates in an unexpected manner or fails to operate at all
- Software (and/or hardware) provisions for:
  - Software written using best practices/coding standard
  - Logical monitoring of the program sequence, coupled with the triggering of the independent monitoring device
  - Inclusion of built-in self-tests or other fault detection mechanisms as required by the Risk Analysis, particularly for the contents of memories (non-volatile or volatile), monitoring for illegal operations (e.g. division by zero), stack monitoring,
  - Clear separation between critical and non-critical sections of software, including privatization of functions and variables dedicated to critical sections, and clear definitions of the interfaces between critical and non-critical sections where and when it is necessary that they interact

Note that the recommendations above are aligned with architectures specified in various functional safety standards, including single-channel with periodic self-test for Class 1/B software in ANSI/UL 1998 and CSA C22.2 No. 0.8, 1oo1D architecture appropriate for SIL 1 in IEC 61508, and Category 2 architecture appropriate for PLs a, b, or c in ISO 13849. Higher Classes, SILs, or PLs may be possible but will require more robust safety measures than the ones described above.

5 Verification and Validation Requirements

5.1 Developer

- Provide evidence that the OTS either:
  - Has been developed under a formal quality assurance program , including design and code reviews, validation testing against a documented set of safety requirements, documentation covering its operation and interface requirements [ref. 2, Clause 5.2.8ci], and verified and tested to the extent that risks involving the OTS software have been addressed [ref. 1, Clause 13.2a], or
  - Has been certified to meet defined requirements by an independent assessor [ref. 1, Clause 13.2b; ref. 2, Clause 5.2.8cii]
- Review known issues or bugs for the OTS and provide evidence that each known issue/bug does not lead to a risk in the end application [ref. 1, Clause 13.3; ref. 2, Clause 5.2.8a]

5.2 ELISA

- Assuming that ELISA is not interested in obtaining third-party certification for Linux, provide a Verification and Validation Report for each distribution, particularly including and verifying the following:
  - What procedures, methods, and criteria were used to develop and test the operating system prior to its official release
  - The results of those tests
- Create Release Notes for each distribution, include lists of known issues/bugs

6 Conclusion

It is proposed that ELISA develop the following deliverables in support of enabling Linux in safety applications that may require compliance with ANSI/UL 1998, CSA C22.2 No. 0.8, or EN 50271:

- Manual, including general information pertaining to the Linux distribution like the version/revision and descriptions of APIs, etc.,

- Fault Analysis, identifying general risks associated with the use of Linux and how those risks have been (or should be) addressed,
- Design Guide, providing recommendations for appropriate hardware and software architectures to use for safety applications
- Verification and Validation Report, describing the procedures, methods, and criteria used to develop and test that Linux distribution prior to its release, and the results of those tests
- Release Notes, describing any known issues or bugs in that particular Linux distribution

The above can be combined into fewer than five documents or expanded into more than five documents as deemed necessary by ELISA.