Linux in Basic Safety Applications

ELISA Group

January 29, 2021

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 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 (hazard 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 and ISO 26262, but still likely require a level of assurance that any embedded software performing safety-related functions will perform correctly when called upon to do so. In these cases, standards such as ANSI/UL 1998 [1], CSA C22.2 No. 0.8 [2], IEC 62304 [3], or EN 50271 [4] are used. These four standards have specific requirements pertaining to third-party software including operating systems.

The remainder of this document summarizes requirements from these standards that would apply to the Linux operating system if it were to be used in a safety-related application. Each section of this document additionally offers pointers to resources that a developer can use to comply with those respective requirements.

While the requirements and resources cited in this document are not as sophisticated or comprehensive as those needed for IEC 61508 or ISO 26262, they have a similar objective to the requirements described in those standards. Therefore, some of what is described in this document could be useful for those standards as well, particularly for lower SILs (SIL 1) or ASILs (ASIL A, B).

1.2 The Standards

ANSI/UL 1998 [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 [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.

IEC 62304 [3] is the International Electrotechnical Commission's standard covering software life cycle processes in medical device software. IEC 62304 calls out requirements specific to Software Of Unknown Provenance (SOUP), which is software already developed and available that was not developed for a specific application, or for which adequate records of the development processes are not available (definition 3.29). SOUP is prominently taken into account in the software risk management process in Clause 7 including risk management of software changes. Note that IEC 62304 requirements cited in this document assume that the software system is considered Software Safety Class B or C, i.e. the software system could contribute to a hazardous situation that could result in injury or death.

EN 50271 [4] 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. 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. Additional clauses of the standard also indirectly impact operating systems.

2 Information

2.1 Requirements Summary

The developer is 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 (UL 1998 [1] Clauses 13.1a, 13.1b; CSA C22.2 No. 0.8 [2] Clauses 5.2.8a and 7.2.4; IEC 62304 [3] Clause 8.1.2; EN 50271 [4] Clause 4.3.4)

- Description of purpose and function of OTS software (UL 1998 [1] Clauses 13.1c, 13.1d; CSA C22.2 No. 0.8 [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 (UL 1998 [1] Clause 13.1e; CSA C22.2 No. 0.8 [2] Clauses 5.2.8b, 5.6.1.1f, and 7.2.4)

- References to OTS software documentation for each callable routine used (UL 1998 [1] Clause 13.1f; CSA C22.2 No. 0.8 [2] Clauses 5.2.8b, 5.6.1.1f, and 7.2.4)

- Configuration and build description of OTS software (UL 1998 [1] Clause 12.4)

- Methods and activities to maintain and control changes made to the OTS software, including its configuration (UL 1998 [1] Clauses 12.4 and 14); this includes procedures to evaluate and implement upgrades, bug fixes, patches, and obsolescence (IEC 62304 [3] Clause 6.1f), which shall include risk management of software changes made to SOUP including impact analysis to determine if changes could contribute to a hazardous situation, could interfere with existing risk control measures, or if additional risk control measures are needed (IEC 62304 [3] Clauses 7.4.1 and 7.4.2)

- Approach and rationale used to select the OTS software (EN 50271 [4] Clause 4.3.5.3.1j)

2.2 Resources

Go through each of the bullet points above and list places where this information could be obtained. In particular, it was mentioned previously that hashes are used to uniquely identify builds.

To do

3 Risk Analysis

(Note: ANSI/UL 1998 [1] and CSA C22.2 No. 0.8 [2] require a Risk Analysis to identify appropriate safety measures and IEC 62304 [3] requires consideration of SOUP in the risk management process, whereas EN 50271 [4] prescribes how risks are addressed by design. While what is required in theory depends on which standards are normative, it is recommended that both this section and the Design section below are considered.)

3.1 Requirements Summary

The developer is responsible for conducting and documenting a Risk Analysis that determines:

- The set of risks (UL 1998 [1] Clause 3.1a; CSA C22.2 No. 0.8 [2] Clause 5.3.1), based on the safety requirements (UL 1998 [1] Clause 3.2), which considers microelectronic hardware faults (UL 1998 [1] Clauses 8.1 and 8.2; CSA C22.2 No. 0.8 [2] Clause 5.3.2.2), software faults caused by microelectronic hardware faults (UL 1998 [1] Clause 8.3), other states or transitions capable of resulting in a risk (UL 1998 [1] Clause 3.4; CSA C22.2 No. 0.8 [2] Clause 5.3.2.1), and the OTS software (CSA C22.2 No. 0.8 [2] Clause C.4.2.2)
- How the software addresses these risks to acceptable levels (UL 1998 [1] Clause 3.1b; CSA C22.2 No. 0.8 [2] Clauses 5.3.2.3 and 5.3.2.4)
- Potential causes of the software item contributing to a hazardous condition; in particular, failure or unexpected results from SOUP (IEC 62304 [3] Clause 5.1.7, 7.1.2c)

Additional considerations should be given in the Risk Analysis for the following:

- Failure of proper allocation of task resources, including scheduling frequency of tasks, criticality of tasks, and resources utilized by the tasks (UL 1998 [1] Clause 6.5; EN 50271 [4] 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) (UL 1998 [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.

3.2 Resources

Go through each of the bullet points above and list places where this information could be obtained. In particular, it was suggested to include recommendations for development tools. Need to also include a caveat that a Risk Analysis cannot be conducted at the operating system level without considering the end application. However, many failure modes of operating systems are known, and these could be considered in the Risk Analysis. Is there a resource available online that we could point to for this?

To do

4 Design

(Also see Note above for Risk Analysis section)

4.1 Requirements Summary

Responsibilities of the developer include:

- Must initialize software and product to a risks-addressed or safe state as appropriate for that product (UL 1998 [1] Clauses 7.1 and 9.2)
- All variables in software shall be set to initial values before being used (UL 1998 [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 (UL 1998 [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 (UL 1998 [1] Clause 6.4; EN 50271 [4] Clause 4.6); run-time checks and built-in tests shall be conducted automatically as appropriate for the application (EN 50271 [4] 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 (EN 50271 [4] Clause 4.6e)
  - Parameter memory and RAM shall be protected against single-bit faults and most double-bit faults, and tested for readability/writability (EN 50271 [4] Clauses 4.6e and 4.6f)
  - Other tests as deemed necessary by the Risk Analysis (UL 1998 [1] Clauses 8.1 and 8.2; CSA C22.2 No. 0.8 [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 (UL 1998 [1] Clause 6.6); more specifically:
  - Product shall transition to a risks-addressed or safe state upon power interruption of any duration (UL 1998 [1] Clause 9.1; EN 50271 [4] 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 (UL 1998 [1] Clauses 6.1-6.3, 7.6; EN 50271 [4] 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 (UL 1998 [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 (UL 1998 [1] Clauses 7.8-7.9, 7.11)

- Control of the execution of the software shall be maintained and ensured (UL 1998 [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 (EN 50271 [4] 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 (EN 50271 [4] Clause 4.3.2.2b)
- It shall not be possible for the user to modify the configuration of the operating system (UL 1998 [1] Clause 10.2, EN 50271 [4] Clause 4.3.2.1b)
- It shall not be possible for the operating system to automatically update (EN 50271 [4] Clause 4.3.2.1c); only the developer shall have the ability to do this, fully under their control (EN 50271 [4] Clause 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 (UL 1998 [1] Clause 7.10; EN 50271 [4] Clause 4.3.2.1e)
- Safety-related input and output ports are read and controlled by the application software; not by the operating system (EN 50271 [4] Clause 4.3.2.2c and 4.3.2.2d)
- Additionally, the requirements for the software system shall include functional and capability requirements that include any need for compatibility with upgrades or multiple SOUP or other device versions (IEC 62304 [3] Clause 5.2.2a), functional and performance requirements for the SOUP item necessary for its intended use (IEC 62304 [3] Clause 5.3.3), including system hardware and software necessary for proper operation of the SOUP item (IEC 62304 [3] Clauses 5.3.4 and 5.3.6)

Potential measures to address the above include the following:

- 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 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 [1] and CSA C22.2 No. 0.8 [2]. It may be possible to achieve higher Classes or even achieve a particular SIL per IEC 61508 or ASIL per ISO 26262, but it will likely require more robust safety measures than the ones described above.

4.2 Resources

Go through list above and provide suggestions for reference designs such as E-GAS.

To do

5 Verification and Validation

5.1 Requirements Summary

Responsibilities of the developer include:

- 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 (CSA C22.2 No. 0.8 [2] Clause 5.2.8ci), and verified and tested to the extent that risks involving the OTS software have been addressed (UL 1998 [1] Clause 13.2a), or
  - Has been certified to meet defined requirements by an independent assessor (UL 1998 [1] Clause 13.2b; CSA C22.2 No. 0.8 [2] Clause 5.2.8cii)
  - Has undergone planning for integration into the software items, including integration testing (IEC 62304 [3] Clause 5.1.5)
- Review known issues (bugs, errata, anomalies, etc.) for the OTS and provide evidence that each known issue does not lead to a risk in the end application (UL 1998 [1] Clause 13.3; CSA C22.2 No. 0.8 [2] Clause 5.2.8a; IEC 62304 [3] Clause 7.1.3)

5.2 Resources

Go through list above. May wish to include recommendations for testing tools. Also consider not only known bugs, but also revision history. What procedures, methods, and criteria are used to develop and test changes to the operating system prior to official release? Where are release notes? To do

References

[1] "UL 1998, Software in Programmable Components," standard, UL, Sept. 2018.

[2] "CSA C22.2 No. 0.8, Safety Functions Incorporating Electronic Technology," standard, CSA, Feb. 2019.

[3] "IEC 62304, Medical Device Software - Software Life Cycle Processes," standard, IEC, June 2015.

[4] "EN 50271, Electrical Apparatus for the Detection and Measurement of Combustible Gases, Toxic Gases or Oxygen - Requirements and Tests for Apparatus Using Software and/or Digital Technologies," standard, EN, June 2018.