Survey on State-of-the-Art Open-Source Operating Systems in Avionics

Martin Halle, Matt Kelly, Samuel R. Thompson, Steven H. VanderLeest

February 2023

Abstract

This paper is a survey of the state-of-the-art for use of Linux in regulated safety-critical systems, especially in aerospace.

A focus on real-time, and a focus on safety-critical.

A focus on Linux, but also looking at other open source operating systems, especially that are unix-like. Also include unix-like operating systems as host OS within a partition hosted by a hypervisor.

Possibly include commercial (closed-source) RTOS as well? As the very least we will overview close-source as context in the introduction.

Comparison Pros/Cons section - does this fit? Instead might be better to identify sensible selection/comparison criteria, quantify where possible.

1 Introduction

This paper surveys the use of open source operating systems in aerospace, with a particular focus on Linux, historically and up to the state-of-the-art.

For systems using Linux, our scope includes utilization of Linux in systems that also include other open-source or closed-source system software, as long as Linux is also used. Our scope covers both air and space, both flight systems and ground support systems. We limit our consideration to systems that have some safety-critical aspect. However, we consider systems not only where Linux itself is supporting the safety-critical function but also where it is present, but not in the safety-critical aspect. For example, the provision of an ARINC 653 environment can isolate a partition containing Linux at a lower criticality level from other partitions at higher criticality levels.

Before proceeding to the main topic, for context we note closed-source solutions and then briefly identify challenges to using open source.

1.1 Closed-Source Solutions

The world of safety-critical Real-Time Operating Systems (RTOSes) and hypervisors is a small one, dominated by a handful of commercial, closed-source players: DDC-I, Green Hills Software, Lynx Software Technologies, Sysgo, and Wind River. These offer relatively expensive solutions that provide assurance evidence suitable to permit their use at the highest levels of criticality. This evidence is typically in the form of a certification package that contains software life cycle data per DO-178C.

While many of these proprietary offerings have been part of aircraft certified up to DO-178C Software Level A (the most critical), open source solutions using Linux have only been flight certified to date up to Level D.

1.2 Challenges to Using Open Source in Safety-Critical Aerospace

At the other end of the cost scale, there are several open-source RTOSes, (see Section 4) that aim to provide similar functionality, with varying levels of success. There are several issues facing the deployment of these open-source projects in an aerospace context. The first challenges is providing adequate assurance evidence for software largely out of the control of the certification applicants or their suppliers. While a commercial RTOS vendor can amortize the cost of preparing a certification package across many expected sales, a user of an open-source RTOS will typically be compelled to bear the cost of these activities alone.

A second challenge facing open-source RTOSes is the absence of an ecosystem, both of engineers with specific expertise in that RTOS and in terms of compatible tooling. This is where Linux-based operating systems could have a significant positive impact: there is already a large community of experts available, and there is a vast ecosystem of compatible tooling. That is not to say that there are not still significant issues preventing widespread adoption of Linux-based RTOSes for highly-critical systems.

The primary aim of this paper is to assess the gap between where Linux is now and where Linux needs to be to become a viable choice for hosting aerospace systems at DO-178C Software Level C and above, in the context of the current state-of-the-art.

2 Motivation

Our motivation in this survey paper is driven by a desire to share in the benefits of the Linux operating system. Linux is widely available and easily adopted because of its open source licensing. This in turn has driven a global collaboration of developers that results in innovation, injection of modern software technologies, broad review, and quick response to issues (including security vulnerabilities). The available workforce with expertise in Linux far exceeds any other pool of operating system developers.

Linux is frequently used as the testbed for investigation, even for avionics topics. Researchers frequently cite the ease of use and availability to test out new concepts while acknowledging that the results must then be translated to another safety-critical operating system. The ubiquitous use of Linux even for this research would greatly benefit if the same operating system used for the research could be used in operation and production.

However, adoption of Linux for safety-critical domains such as aerospace is a daunting challenge. We thus are further motivated in this paper to (1) identify the state-of-the-art so the industry can build on it and (2) identify gaps so the industry can address them.

3 Linux in Aerospace

This section surveys systems that use Linux in aerospace. It is organized into two subsections. The first reviews work to use Linux in safety-critical functions. The second subsection reviews safety-critical systems that use Linux, but not for safety-critical aspects.

This paper does not cover use of Linux as the basis of development tools or as a development environment. We also do not cover early prototyping of flight software that uses Linux at first but then moves to a different operating system in order to flight certify. We focus only on Linux running on computational platforms onboard a flying aircraft.

TBD: Kernel Level Arinc653 partitioning in Linux [**han2012**] and then Covers problems with Linux for avionics in related works [**bloom2020**].

NASA has conducted multiple flight tests using Linux. The Ascent Abort 2 (AA-2) Simulation used three redundant Linux computers communicating over TTEthernet at 200Hz, and carried out flight tests in 2015 at NASA Johnson Space Center. The Orion Exploration Missions 1 and 2's Camera Controller Units of the Crew Module run Ubuntu Linux [**lorraine2018orion**]. The Space and Missile Systems Center's Space Test Program Satellite-4 (STPSat-4) was a small satellite launched from the ISS in 2020 that served as a testbed for payloads developed by developed by the Air Force, the Navy, DoD and academic institutions. STPSAT-4 reported ran Suse Linux for command and data handling, payload data handling, recording, file downlink and on-board autonomy functions [**prokop2015aes**].

The Ingenuity Mars Helicopter, a 1.8kg rotocopter sent by NASA JPL to Mars as a payload on the Perseverance Rover, contained two processors, including a Linux-based Snapdragon processor [**canham2022**]. Throughout the duration of this mission, the helicopter completed 72 flights on the surface of Mars. The Linux-based processor performed control and data handling functions as part of the guidance, navigation and control (GNC) of the aircraft. Images were captured at 30 FPS, and used for analysis by the GNC algorithm, running at at 500 FPS.

SpaceX has reported that it runs Linux on Falcon launch vehicles, Dragon capsules, and Starlink satellites [**spacexama2020**]. The company has publicly stated that they maintain a copy of the kernel with the `PREEMPT_RT` patch applied to get better real-time performance, and their work is not based on an existing Linux distribution. Each Starlink launch of 60 satellites carries with it 4000 linux computers (for a measure of relative size, the number of operational Starlink satellites as of the time of this writing exceeds 5400 satellites).

3.1 Safety-Critical Linux

This section reviews research into Linux supporting safety-critical functions directly, requiring high levels of assurance for the operating system.

There are few research papers that address Linux for aerospace. For example, a 2023 paper [**vanderleest2023avionics**] reports that scholarly papers with both the words "Linux" and "DO-178C" only appear 529 times in Google Scholar and not at all in IEEE Xplore.

3.1.1 Paper Studies

This section surveys research that analyzes Linux for use in aerospace supporting safety-critical functionality. However, the papers in this study do not cite actual operational examples.

A 2004 Presentation by Airbus regarding Linux for civil aviation [**goiffon2004**] identifies key challenges to making Linux ready for certification (at the time, DO-178B). However, they noted the heavily process-oriented standard might not sufficiently recognize the reliability of the modular architecture, the cooperative development model with centralized version management, and extensive product reviews and testing by peers. They foresaw that the way forward would require reverse-engineering of code, validation against standards and for robustness, analysis of Worst-Case Execution Time (WCET) and stack usage, development of a configuration to provide static allocation but dynamic control of resources, and use of robust spatial and temporal partitioning.

Research by Han and Jin [**han2012**] investigated implementation of the ARINC 653 partitioning standard within the Linux kernel. They found that performance was sufficient for typical real-time needs in aerospace. However, they did not analyze certification considerations. A 2018 study [**liu2018**] also investigated real-time performance of Linux, considering three different real-time approaches that augment Linux using a set of patches: Preempt_RT, Xenomai, and RTAI. They concluded that real-time performance was sufficient for aerospace needs, though they pointed to different use-cases appropriate for each patch.

A 2013 prototype project to extend Linux with real-time and safety properties [**cotroneo2013fault**] claimed that their FIN.X-RTOS was compliant with DO-178B Software Level D and they next hope to work toward Level C. While work seems to have continued on the product, the focus has turned to security and no more recent work has been published on their pursuit of Software Level C for safety.

3.1.2 Operational Studies

The papers surveyed in this section cite use of Linux for safety-critical functions of operational missions.

A survey of Linux in space [**leppinen2017**] covers both general purpose computing as well as real-time implementations of Linux in operational space vehicles. The summarize the factors driving use of Linux as increasing complexity of requirements, increasing computing power reducing the need for specialized software, and availability of skilled labor familiar with Linux-based development. At the same time, the paper recognizes challenges to adoption of Linux: 1) requirement for processors with a Memory Management Unit (MMU), 2) the quality assurance of code through verification and validation required for code that might not be needed but is included implicitly through use of software packages, and 3) need for real-time performance.

In 2003 Boeing noted the use of Linux in an Electronic Flight Bag [**allen2003efb**] certified under DO-178B, but did not specify to what software level. In a later article [**allen2008electronic**] they note ruggedized Class 2 units are available through their subsidiary, Jeppesen.

While there are several known uses of Linux in operation at Software Level D, only one published instance is known for Level C, reported by Avionics International [**adams2015**]. They note an Electronic Flight Bag implementation developed by Astronautics.

3.2 Linux Separated from Safety-Critical

This section reviews research into Linux supporting non-safety-critical functions within an aerospace system that also supports safety-critical functionality. Thus, high levels of assurance for Linux are not required, provided it can be isolated. We organize the literature into two subsections. The first reviews Linux used on a computing device that does not have any safety-critical function. The second reviews Linux used on a computing device that has safety-critical functionality, but where Linux is isolated.

3.2.1 Linux in Federated Systems

Traditional avionics systems segregated functionality into separate computing platforms, sometimes known as Line Replaceabable Units. Thus the safety-critical software ran on one computer but non-safety-critical on another. Examples of this approach are found in Macfas [**macias2013open**], where PikeOS is used on a Leon3 processor for safety-critical functions and Linux is used on an ARM9 processor for non-safety-critical functions and in Torens [**torens2014certification**]. a study of Autonomous Unmanned Aircraft where the flight control function is performed by an Intel P4 processor running the QNX operating system and the image processing function is performed by a different Intel P4 processor running Linux.

3.2.2 Linux in Integrated but Partitioned Systems

In the last decade there have been a number of attempts to use partitioning to incorporate the broad functionality and commonly understood interfaces of Linux into aerospace. Partitioning isolates the Linux operating system and its applications, avoiding the high cost of certifying that partition to high levels.

Obermaisser [**obermaisser2008temporal**] discusses isolation of a real-time variant of Linux using time triggering, though they only obliquely cite ARINC 653 as a partitioning mechanism. A 2009 paper[**craveiro2009**],

a project sponsored by the European Space Agency examined the ability of ARINC 653 systems to isolate heterogeneous operating systems, separating the broad functionality of Linux from the safety-critical functionality supported by an assured real-time operating system in another partition. They customized a distribution of Linux, called Bullet Linux, to provide key features while using a smaller system library.

One approach is to use an open source approach to partitioning. VanderLeest [**vanderleest2010arinc**] presented an initial prototype of an ARINC 653 hypervisor based on open source Xen, with Linux as a guest operating system in a partition. The paper provides an analysis of sharing resources (CPU, memory, and I/O) between partitions. Cofer [**cofer2022cyberassured**] describes a prototype using the seL4 microkernel that separates Linux into a guest virtual machine for use on a Unmanned Air Vehicle.

Another approach is to use a closed source approach to partitioning. Several proprietary vendors offer a partitioning approach (typically ARINC 653), including [**lim2021autonomous**] using Qplus-AIR.

The security community has used a similar concept to isolate Linux using a small, assured hypervisor or microkernel. A team at Carnegie Mellon[**de2019mixed**] termed this "disjoint-trust computing".

4 Open Source Non-Linux RTOS in Aerospace

Furthermore to consider: JetOS [**jetos2016**] which is a fork of POK [**pok2013**], FreeRTOS [**freertos2023**], Zephyr [**zephyr2023**], RTEMS ([**rtems2023**]), NuttX ([**nuttx2023**]).

4.1 JetOS

Two promising OS RTOSs claim to be fully compatible with ARINC 653: JetOS [**jetos2016**] and POK [**pok2013**], which are being developed in Russia and France respectively. JetOS is a real-time operating system for avionics applications, which originates from the open source project POK. JetOS is being developed at ISPRAS (Ivannikov Institute for System Programming of the Russian Academy of Sciences) in the scope of the research and development project. It is released under the GPLv3 license. The source code is publicly available on GitHub. JetOS is intended primarily for onboard equipment based on the integrated modular avionics. This RTOS can be run on PowerPC and x86 CPU architectures. Configuration of the system is stored in XML documents. This approach eliminated the necessity of the AADL (Architecture Analysis and Design Language) configuration tool Ocarina, which is supported by the POK RTOS.

4.2 POK

POK [**pok2013**] is a real-time embedded operating system intended for use in safety-critical systems. POK is released under the BSD license and has been used in several research projects, one of which is JetOS mentioned previously. The source code is publicly available on GitHub. POK supports several architectures, among which are x86, PowerPC, Sparc/LEON. POK has two layers: Kernel and partition. Kernel-layer services are executed at high-privileged level (kernel mode). Whereas partition-level services, which support applications execution, are executed at low-privileged level (user mode). POK uses the AADL tool-suite Ocarina to specify system architecture and to define its properties, and to generate the configuration and runtime code. Unfortunately neither JetOS nor POK seem to have a developer-community support. In case of POK community support is done through GitHub. But it showed itself to be unreliable. JetOS has no community support at all. Furthermore JetOS does not have sufficient documentation.

4.3 FreeRTOS

FreeRTOS [**freertos2023**] is an open-source real-time kernel designed specifically for micro-controllers and small microprocessors. Apart from a kernel FreeRTOS also includes a growing set of IoT libraries suitable for use across all industry sectors. The key features of FreeRTOS are reliability, accessibility and ease of use. FreeRTOS is owned, developed and maintained by Real Time Engineers Ltd. and is distributed freely under the MIT open source license. It has been ported to several different architectures, including ARM, Intel x86 and PowerPC, etc. FreeRTOS has a migration path to another real-time operating system - SafeRTOS [**safertos2023**], which includes certifications for the medical, automotive and industrial sectors. FreeRTOS kernel consists of only a few source files, which must be included in the application project. Its API is designed to be simple and intuitive and it partially supports POSIX through POSIX threading wrapper. FreeRTOS is fully supported and exhaustively documented. FreeRTOS has no memory protection in general, but it provides official Memory Protection Unit (MPU) support on ARMv7-M and ARMv8-M cores. To protect critical regions FreeRTOS uses recursive mutexes with priority inheritance.

4.4 Zephyr

Zephyr [**zephyr2023**] is a scalable, real-time operating system for embedded devices. It is developed with safety and security in mind. Zephyr is available through the Apache 2.0 open source license, which means it can be used in commercial and non-commercial solutions. Zephyr supports multiple architectures, including x86. Community support is provided via mailing lists and Slack. The users of Zephyr are provided with an exhaustive documentation and multiple demo code examples to start with.

4.5 RTEMS

RTEMS (Real-Time Executive for Multiprocessor Systems, [**rtems2023**]) is an open-source real-time operating system targeted towards deeply embedded systems. It is competitive with proprietary products and is used in space flight, medical, networking and many more embedded systems with strict timeliness requirements. In addition to the native API RTEMS also supports open standard application programming interfaces (API) such as POSIX. Among processor architectures supported by RTEMS are ARM, PowerPC, Intel, SPARC, MIPS, and more. RTEMS is distributed under a modified GNU General Public License (GPL), which means the source code is available for use, tailoring and redistribution. RTEMS provides an exhaustive documentation and has a supportive user community. There exist several projects, the goal of which was to run RTEMS on ARINC 653 compliant POK. The both projects implemented the concept of paravirtualization with POK being a hypervisor and RTEMS - a guest operating system. According to the final reports of both projects there were still some problems, though some parts of the projects were successfully completed. Another attempt to make RTEMS compliant with ARINC 653 was made in scope of an ESA (European Space Agency) project named AIR [**air2007**]. Its final report provides a detailed analysis of the AIR design solutions. It presents the workflow of implementation of APEX services on top of RTEMS API and defines the issues for further full implementation of APEX.

4.6 NuttX

NuttX [**nuttx2023**] is a real-time operating system with an emphasis on standards compliance. The primary governing standards are POSIX and ANSI standards. Additionally standard Unix APIs and APIs of other common RTOSs (such as VxWorks) are adopted in order to provide functionality not available under POSIX and ANSI standards. To the supported platforms among others belong x86 and MIPS. NuttX is released under non-restrictive Apache license. NuttX is supplied with a well documented user guide. Community support is provided via mailing lists.

chibios Used in ArduPilot: https://www.chibios.org/dokuwiki/doku.php NASA Langley has used ArduPilot

5 Comparison, Pros and Cons

5.1 Comparison Criteria

5.1.1 Availability of partitioning/timing determinism features

Some RTOSes will provide software-based partitioning mechanisms which can be used to partition resources and promote timing determinism. Some of these (e.g. Deos has a memory pooling feature which allows cache lines to be reserved for certain tasks) are applicable to virtually all embedded computing platforms, while others (e.g. Green Hills Integrity has an interconnect bandwidth limiter) are only applicable to multicore systems.

For any safety-critical RTOS, the availability of robust mechanisms that enhance determinism can be a powerful tool for eking the best throughput from a platform; therefore this is an important comparison criteria.

5.1.2 Available Capabilities

Availability of RTOS features is another key metric for comparison. Some OSes (e.g. most Linux distributions) will come with large repositories of packages giving a vast array of functionality. At the other end of the scale, some RTOSes have the bare minimum hardware support to provide a minimal partitioned runtime environment – a simpler software ecosystem will typically lend itself to easier/faster/cheaper certification.

Clearly there is a trade-off to be made here between a simple system that is more easily certified and a more complex system that provides many (possibly useful, or even essential) features.

5.1.3 Hardware Support

Some RTOSes are very tightly coupled to a single architecture, while others will support being built/executed on a range of different architectures. Moreover, some RTOSes will come with wide-ranging driver support for a number of types of peripheral, while others will leave provision of drivers to the end user.

This can have a significant impact on the usability of RTOSes (and indeed the skillset requires to effectively develop applications on), so should be considered a key comparison criterion.

5.1.4 Software Development Tools

Engineers working on software will usually spend a lot more time using the development tools provided with an RTOS than they will spend working directly on the RTOS. That is, it doesn't matter how highly-polished an RTOS itself is if the tooling ecosystem is hopelessly immature and buggy – a poor ecosystem will result in frustrated developers and wasted time.

The software development tools in this context will include the build system, the configuration system, the debugging tools (if any), and frequently an IDE that integrates everything together. In many cases, it is simply not possible to use the RTOS without the included tooling. Therefore, the included tooling should be evaluated at the same time as the RTOS itself.

5.1.5 Technical Support

When purchasing licences to a commercial RTOS, it is common practice to also purchase some number of support hours from the vendor. Depending on the vendor, this may be a positive experience, or the support may be found inadequate, slow, or otherwise displeasing. For an open-source OS, it is certainly possible to make use of it without any support contract in place, though for some open-source OSes there are vendors who provide technical support on a commercial basis.

In order to make effective use of limited engineering time, timely and high-quality support can be a significant differentiator for any RTOS, open-source or not.

5.1.6 Certification Support/Certifiability

Fundamentally, if an RTOS is to be usable in a safety-critical context in the aerospace industry, it needs to at the very least be certifiable. Commercial RTOS vendors will typically sell 'cert packs', which contain all the necessary certification material for their RTOS. Typically, this will only support certain configurations or types of deployment of that OS, and may preclude the use of certain features.

The availability of such certification support material can represent a very significant saving of time on the part of the certification applicant, at the expense of the cost of the certification pack (which is typically not cheap).

For a meaningful comparison between RTOSes, therefore, the availability of a certification pack of some kind (or at least, materials that support a certification effort) is a crucial metric of comparison.

5.2 Comparison

6 Summary

Many papers considering Linux for aerospace applications focus on real-time performance requirements. Many of them identify various means of adjustment or customization that enable a Linux implementation to meet these requirements. In summary, it appears that a properly configuration and tuned Linux operating system is suitable for most real-time aerospace use cases.

Most of the papers considering Linux for aerospace applications completely ignore certification issues. Those that do mention the quality assurance effort via validation and verification do not recognize the colossal effort this represents. It is our conclusion that despite all the focus on real-time performance, this is not the main barrier to adoption of Linux in aerospace. Rather, the primary obstacle is the high cost of safety assurance. This is borne out by the more prevalent use of Linux in space and military use-cases compared to the the dearth of published examples for commercial flight, particularly at higher DO-178C software levels corresponding to higher criticality requirements.

6.1 Safety-Critical Linux in Other Industries

The research literature also contains some mentions of Linux in other safety-critical industries.

Procopio [**procopio2020safety**] considers Linux for use up to Safety Integrity Level (SIL) 2 in IEC 61508 (industrial) systems. SIL 4 is the highest criticality.