Scientific Research Publishing

Int. J. Communications, Network and System Sciences, 2025, 18(2), 15-25
https://www.scirp.org/journal/ijcns
ISSN Online: 1913-3723
ISSN Print: 1913-3715

Unix-Based Systems in Embedded and IoT Devices: Exploring the Versatility and Robustness

Hitendra Chaudhary[1], Andrew Mikhl Anthony[1], Olatunde Abiona[1], Clement Onime[2]

[1]Department of Computer Information Systems, Indiana University Northwest, Gary, IN, USA
[2]Information & Communication Technology Section, International Center for Theoretical Physics, Trieste, Italy
Email: hchaudha@iu.edu, andanth@iu.edu, oabiona@iu.edu, onime@ictp.it

How to cite this paper: Chaudhary, H., Anthony, A.M., Abiona, O. and Onime, C. (2025) Unix-Based Systems in Embedded and IoT Devices: Exploring the Versatility and Robustness. *Int. J. Communications, Network and System Sciences*, **18**, 15-25.
https://doi.org/10.4236/ijcns.2025.182002

Received: April 23, 2024
Accepted: February 25, 2025
Published: February 28, 2025

Abstract

The Internet of Things (IoT) has transformed the relationship between people and systems. Many IoT use embedded systems even for specialized applications. The purpose of the research paper is to explore the importance and role of Unix-based operating systems for embedded and IoT systems. In this paper, we explore the versatility and robustness of a Unix-based system, with its standby issues; we also discuss the overall security, real-time performance, cloud and edge integration, containerization, and power energy problems. This paper delves into the different methods that provide security, such as the integration between the cloud and the edge, virtualization, and energy consumption of Unix-based systems for embedded systems related to IoT applications. It also examines the current frameworks and enabling tools of the Unix-based system, which covers a variety of frameworks and different tools. The conclusion of the research paper is that IoT systems must take advantage of all the services based on Unix-like operating systems. Ultimately, the proposed work considered Unix-based systems in embedded and IoT devices as suitable candidates for the future of embedded and IoT systems in limiting and improving areas.

Keywords

Unix-Based Systems, Embedded Systems, Internet of Things (IoT), Security, Real-Time Performance, Cloud Integration, Edge Computing, Containerization, Virtualization, Energy Efficiency, Development Tools

1. Introduction

An innovative idea that changes the way humans interact with systems and gadgets in our environment, the Internet of Things (IoT), came into being during the

Internet era. Embedded systems, which are computer systems built for specific tasks and embedded in bigger devices, make up many of these gadgets. Embedded systems are the basis of most current Internet of Things (IoT) products and gadgets, including smart refrigerators, thermostats, wearables, and even radios. Concurrently, it can also refer to more intricate systems like industrial control systems or electrical systems in automobiles. Recently, The International Center for Theoretical Physics (ICTP) developed an application that used IoT sensors to monitor the quality of the drinking water from a river or stream in a developing nation, with the aim of collecting the data set and using Artificial Intelligence (AI) to predict water quality in the future.

There is an ever-increasing need for embedded systems that are quicker, stronger, and more secure. Given the specifics of the issue, developers often gravitate toward Unix-based operating systems. Systems that are based on the Unix philosophy have been around for a long time; the first version came out in the 1960s, and it has a proven architecture. Reliability, security, scalability, and modularity are four important qualities of an embedded system, and Unix-based OSs are known for all four [1]. This article explores the repurposing of Unix-based Operating Systems for use in embedded and IoT systems, discussing their advantages, disadvantages, practical uses, and prospects.

2. Literature Review

The Unix-based systems have evolved in the embedded and IoT devices as an answer to the limitations of proprietary or custom operating systems for their particular hardware development. The embedded systems have turned to Unix-like OS due to growing complexity and standard-driven demands for embedded software, as well as challenges for cross-platform development. Comer and Shaughnessy wrote an article about it all in 1997 "Unix and Embedded Systems" and stated that Unix or Unix-like embeddable systems featured a number of considerable benefits. In the article, regarding the embedded systems and Unix, they mentioned a great scalability feature: "Unix scales with the needs of the designer" [1]. It makes the development of software for hardware that supports the OS more of a flexible and scalable process for developers.

Satyanarayanan's work led to the development of Unix-like operating systems, which today are used in the Internet of Things or IoT industry when embedded devices can be connected to the Internet and controlled, monitored, and when necessary, data can be shared remotely [2]. This industry's growth spurred interest in various problems in embedded and IoT systems that run Unix. According to Corbet *et al*., the requirements for real-time performance in the latter application like industrial control systems and robotics were explored, and it was determined that the performance of Linux as a real-time operating system or RTOS can be improved [3]. Specifically, the authors argued that time-critical apps that have used a Unix-like OS require deterministic behavior and low-latency response time.

Several studies have focused on Unix-based systems and their security features

and methodologies because security is a major problem in the IoT arena. Zaddach *et al*. examined the security architecture of Linux-based IoT devices and identified vulnerabilities and potential solutions. They found that secure communication protocols, kernel hardening, and secure boot processes were the most important ways to protect IoT devices from hackers and ensure the confidentiality and authenticity of data [4]. Over the past few years, the embedded and IoT industries have experienced a dramatic increase in the development of virtualization and containerization technologies for Unix-based systems. Morabito *et al*. identify some of the many reasons for using the containers in IoT applications, such as lightweight virtualization, efficient resource usage, and the easier deployment and maintenance of applications.

Working with Unix-like operating systems, the authors investigate how containerization could ameliorate some of the heterogeneity and resource limitations of IoT devices. New horizons are opened, and old problems are resolved with the burgeoning use of Unix-based systems in embedded and IoT applications. Improving energy efficiency, bolstering security to handle new kinds of attacks, and simplifying integration with cloud and edge computing systems are priorities. The IoT technology aims to unify everything in our world under a common infrastructure, giving us not only control of things around us, but also keeping us informed of the state of the things [5].

Figure 1. Types of embedded OS.

Figure 1 above depicts types of embedded Operating Systems (OSs), which are integral to powering dedicated non-computer devices. There are multitasking OS like Linux that can handle several processes simultaneously, ideal for complex systems. Real-Time Operating Systems (RTOSs) are designed for tasks requiring immediate response, which is vital for safety-critical applications. Single-task OS like Android manages one operation at a time and is used for simpler devices. Lastly, Rate Monotonic OS prioritizes tasks by frequency, ensuring timely execution for

regular, cyclical operations.

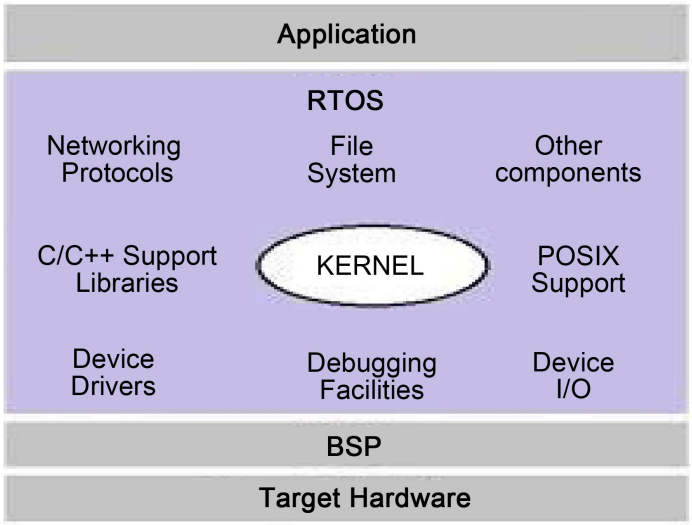

Figure 2. Architecture.

Figure 2 above represents the illustration of an embedded system architecture. The very heart of the image is the kernel of Real-Time Operating System, while around it, there are support elements: C/C++ libraries, device drivers, debugging facilities. All of which are interfaced to the Board Support Package (BSP), which interacts directly with the target hardware. Further, there is an application layer on top of the RTOS that is designed to manage networking, file systems, etc. In other words, it is responsible for making the embedded system run dedicated applications.

Connected embedded or IoT systems typically implement some form of over-the-air update procedure, which requires robust checks, to avoid installing malicious or untrustworthy updates to the system. Other securities mechanisms such as Trusted Platform Modules (TPMs) or similar hardware security modules, facilitate secure storage of keys as well as carrying out cryptographic operations, and attestation capabilities in hardware contribute to the security of the embedded or IoT system.

Figure 3 illustrates the concept of edge computing within the IoT. As shown, IoT devices embedded in sensor objects, including cars, buses, solar panels, and industrial machinery all connected to local edge computing nodes. These nodes perform some data analysis close to the data source, in near real time, before resulting data is transmitted to the central cloud or remote data center. Edge computing improves efficiency as the work is offloaded from the central servers, and the transmission times are reduced.

Furthermore, it is also possible for the developers to use the cloud-based platform to design simple control interfaces, dashboards, and visualization for the easy monitoring and control of their embedded or IoT devices. Through such interfaces, the status, performance, sensor and other information, and settings can

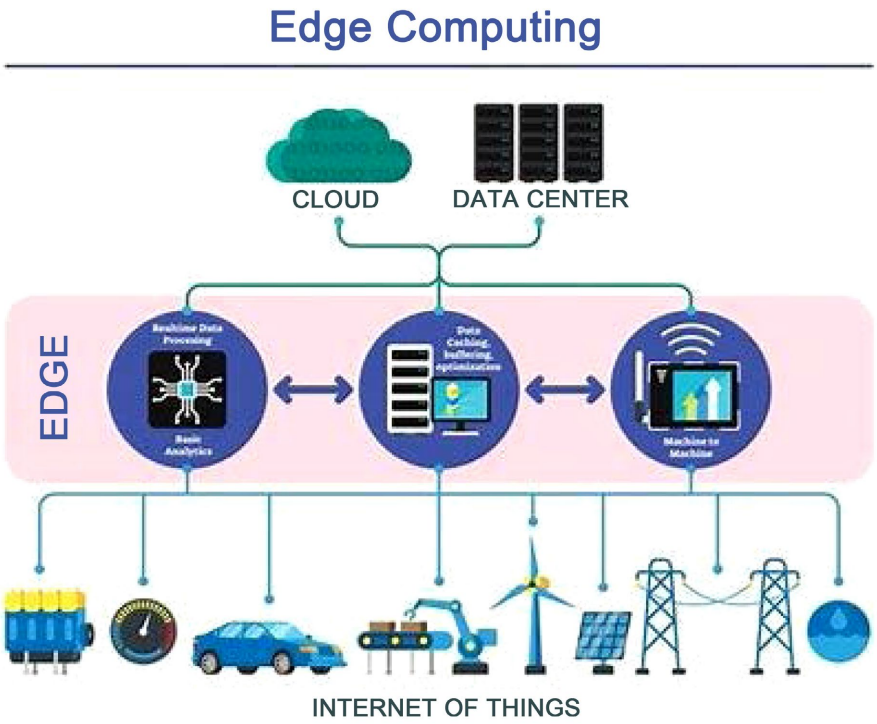

Figure 3. Integration with cloud and edge computing.

be monitored. Moreover, the edge node equipment can analyze the information and make some decisions with the help of information obtained from the other equipment also connected to cloud, thus, reducing the latency and bandwidth.

Containerization and Virtualization Support: Madakam *et al*. suggest the use of containerization and virtualization in embedded systems and IoT. Implementation requires lightweight runtimes environment as well as Virtual Machines (VMs), ensuring refined resource usage, separation of different environments, and broader solution scalability. In this solution, the kernel layer is optimized and adapted to the specifics of container technology such as Docker and Kubernetes. Standard container software development tools can be used to manage the deployment, scaling, and resource separation aspects of packing apps into lightweight containers for embedded and IoT systems.

The virtualization layer is useful for lightweight virtualization and not full virtualization as required by hypervisors such as Xen and KVM. Such a layer allows for the construction of safe and isolated spaces where several different operating systems or applications can be held and run at the same time [6]. This comes in handy when apps must rely on each other, yet they cannot coexist, or they must be segregated owing to health, legal, or public policy requirements. The virtualization layer offers paravirtualization and hardware-assisted virtualization to ensure efficiency in embedded systems with minimal capabilities. Suitable tools may also assist in the virtual machine management process as well as snapshot creation and live migration procedures.

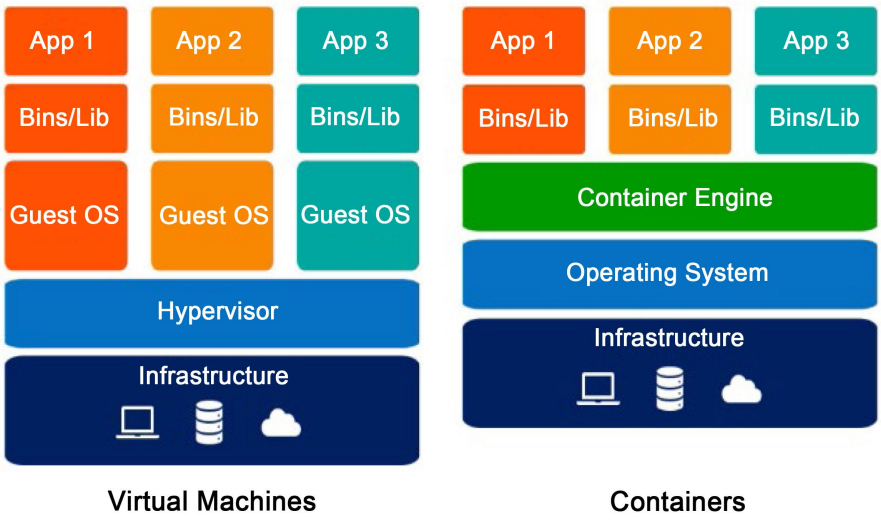

Figure 4. Containerization and virtualization support.

Figure 4 above shows a comparison of virtual machine virtualization and containerization. The main difference between the two is that VMs have their individual dedicated guest operating system above the hypervisor and the rest of the infrastructure, which is isolated but requires many resources. Containers use the OSI kernel managed by the container engine which is lightweight. VMs and Containers execute the applications using their binaries and libraries, however, Containers use resources more effectively.

3. Versatility and Robustness of Unix-Based Systems

Based on the knowledge gained from previous research and the problems that Unix-based systems faced within embedded and IoT applications, this section of the paper examines the use of Unix-like OSs for embedded and IoT systems. While also discussing how to address the common drawbacks of Unix-like OSs. The goal is to form a sound, secure, and high-performance basis for the subsequent development of embedded and IoT systems using Unix-like OSs. This means that the proposed approach can facilitate the effective development of various systems that account for real-time performance, security issues, resource efficiency, and staying in cadence with other emerging technologies [7]. Flexible and Simplified Architecture Unix-based systems are notorious for their modular structure, which allows developers to adjust the OS according to specific hardware configurations and user needs. In contrast to employing a single modular OS structure, Unix-like OSs allow embedded and IoT systems to benefit from the composition layered approach to the design of systems.

Many embedded and IoT devices rely on a slim, hardened Linux kernel, as most critical functions are running within the kernel. Indeed, the emphasis is on the security and efficiency of the designed system. To enhance the security and resistance to the attacks that commonly involve bypasses and code injections, several mechanisms such as Data Execution Prevention and Position Independent Executables

have been implemented by the kernel, as well as immutable access control, secure booting, and Kernel Address Space Layout Randomization (KASLR).

The middleware layer described is modular and located over the kernel. This layer employs a set of reused components and libraries, which are domain-specific and developed for the needs of multiple applications. Those domains include home automation, industrial automation, automotive systems, and consumer electronics [8]. Consequently, a lower application overhead has been achieved through the modular layered form of the middleware, combined with selective inclusion of provided components and exclusion of those that are not useful for the application. A final measure used in the middleware layer is the integrated security framework, which is an engine that executes cryptographic functions, techniques and supported communication protocols.

The highest level is the application layer, where developers can create and deploy their embedded or IoT apps using provided frameworks and APIs [9]. Such separation of concerns makes the code more reusable and thus easier to maintain, reducing the complexity of development. Application deployment, management, and monitoring utilities and tools are integrated into the application layer for a hassle-free operational and development experience.

Steps to Enhance Safety: One of the significant concerns that is associated with embedded and IoT devices is improving their safety, as these devices operate valuable systems, control important infrastructure, or handle sensitive data. Any architecture that enforces an extensive and layered security strategy provides an answer to this issue efficiently.

Hardened Linux kernel provides advanced security features by using security measures such as KASLR, secure boot procedures, MAC mechanisms, etc., to prevent attacks and solve vulnerabilities. Secure boot process ensures the possibility to run only verified and trusted code during the boot process to avoid any tampering and addition of malicious codes. The middleware layer includes a security framework, which helps to manage both cryptographic operations and access to secure communication protocols and security policies. This security framework uses secure communication protocols and the industry's accepted encryption schemes to ensure that data remain secure and confidential during their transmission and storage.

Embedded and IoT devices can leverage the middleware-integrated security framework for secure over-the-air updates/upgrades. This method is important because software upgrading is crucial for deploying the fixes, security vulnerabilities being patched, and designing systems lifetime support [10].

Constraints on Time and Accuracy: Deterministic behavior and real-time performance are common requirements for various embedded and IoT applications, such as safety-critical systems, industrial control systems, and others. The Linux Kernel can be extended to provide a comprehensive environment for real-time application development. Currently, with the PREEMPT_RT patch added to the base kernel layer, it becomes possible to have a fully preemptible kernel and min-

imum-latency response times. In general, this patch can enhance the real-time capacities of the Linux kernel, making them more responsive with lower latencies. It can be vital for applications that rely on exact time [11]. The PREEMPT_RT patch can help speed up the Linux kernel to make it more responsive to real-time applications.

Moreover, the PREEMPT_RT patch provides a real-time framework in the middleware layer that comprises APIs and tools for developing and testing real-time applications. More specifically, this allows developers to use techniques that make the behavior of the system predictable and deterministic. Those techniques include priority inheritance, rate-monotonic scheduling, and resource access control. In addition, the middleware includes profiling, debugging, and performance analysis tools. In such a way, developers can optimize their real-time applications and meet timing requirements. Techniques that are used to boost performance in resource-constraint scenarios include real-time task scheduling, handling interrupts effectively, and priority-based resource management. The use of these techniques will enable scheduling and allocating resources to the most critical real-time tasks and, at the same time, reduce latencies and jitter.

Integration with Cloud and Edge Computing: The Internet of Things was made more significant and incorporated into our lives as the development of IoT devices and their use with cloud and edge computing. The integrated security framework provides a thin and protected communications layer enabling remote control, monitoring, analysis, and data sharing. Therefore, the embedded devices can use it to communicate with each other within a few seconds [12].

Moreover, the data can be effectively transferred from the embedded devices to cloud or edge computing, and, as a result, the variety of Internet of Things related application and services can be found. The implementation has involved, for example, such things as secure authentication, encryption, and data integrity verification, and some API of the implementation can support many communication protocols [13]. Also, many libraries and tools have been implemented over the integrated security framework to make the data marshalling and unmarshalling apparent. It can be used with various data formats, such as Protocol Buffers, FlatBuffers, and JSON. For instance, it is possible to easily reconfigure it to deal with the low-power device constraints and reduce the processing overhead and bandwidth demands.

Development Tools and Ecosystem: Unix-based OSs take advantage of the vast ecosystem of development tools, libraries, and configurations that are available in Unix-based systems. For example, Linux (a Unix-based), systems contain hundreds of tools and utilities that make development processes much easier. They are also well known for their power and being an open-architecture system. IoT and embedded systems development framework build on this rich ecosystem to provide a unified set of development tools and utilities that simplify development and save developers' time. For example, developers can use a proper Integrated Development Environment (IDE) for their embedded and IoT applications [14]. In Linux, several IDEs focus on cross-platform development, allowing developers to write code that can run on various types of hardware architectures and operating systems.

IDE facilitates application development and testing by providing tools for text editing, debugging, and profiling: the complexity of the device makes most of the application development tools unsuitable. In addition, the framework ships with a set of libraries and frameworks that can be used for networking, device drivers, sensor integration, data processing, machine learning, etc.—all the usual tasks in the embedded and IoT industry. Designed for resource efficiency, these libraries comply with all rules of performance security and best practice in the industry. A hardware emulator/simulator supports rapid prototyping of the system, leading to a fast development process [15]. Software engineers no longer need to worry about cost and time of buying the dedicated environment: they can create a virtualized environment to test. the system and the application that runs on top of it. For the development and testing of an application, that facilitating function uses capabilities for automated testing and integrated and continuous deployment.

Energy Efficiency and Power Management: Most embedded and Internet-of-Things devices are battery or solar-powered since they can function in areas that lack access to conventional power sources. As a result, the framework includes designs for power management and energy-efficient ways of preventing the battery from dying as quickly. Power management methods are available at the kernel level. Intelligent task scheduling, S3 and S4 modes, and dynamic CPU frequency scaling are examples. The system can vary its energy consumption based on the workload and the resources required to complete the task due to these methods and implementations. As a result, it does not overuse power. Additional power management tools for controlling energy consumption are also implemented at the middleware level. Developers will be able to create apps that adhere to energy-saving standards using the included APIs and libraries. An example of this is duty cycling, which the developer can use when power is needed on an on-demand basis to flip power-hungry components on and off. Finally, hardware developers prefer low-power components and power-aware scheduling.

Moreover, alternative energy sources such as solar panels, thermoelectric generators, and kinetic energy harvesters are easily integrated. You can use either of these techniques for charging the battery or even replacing it in some IoT devices. To enhance the energy-efficiency, we can also implement Machine Learning (ML) and Artificial Intelligence (AI) frameworks and APIs for workload prediction and resource optimization. The system considers all available data and usage trends, so that it could predict a workload and decide which components would most probably be used.

4. Conclusions

The flexibility, robustness, and open-source nature have made Unix/Linux a preferred choice OS for developers as well as corporations. As we focus on IoT and AI, Unix/Linux role becomes increasingly very important. Unix-based systems are a viable and dependable solution for embedded and IoT devices. Developing such systems may have both benefits and drawbacks regarding their characteristics in some areas, such as security, reliability, scalability, and performance.

Embedded and IoT systems running Unix-like operating systems have unique challenges, which were discussed and addressed in this paper. The Unix-based OSs are highly configurable and flexible to adapt to embedded and IoT environments, as well as to address security issues and propose methods of higher energy-aware design, implemented from a security-oriented standpoint. The modular design enables users to create systems that work with their hardware and improves performance optimization and resource management [16]. There are several security mechanisms and system architectures at different levels available, and they are highly configurable.

A great number of embedded applications depend primarily on the system's built-in determinism technology and the real-time operating system to ensure the system's activity is aligned with definitions and reliably predictable all the time. Furthermore, the technology presents an exciting possibility of administering applications in the cloud and on the edge, providing a seamless and IoT application-focused approach.

Every application is always isolated, meaning that each program can signal the outside world as needed, including sending and receiving data. With virtualization systems, it is possible to install several applications in multiple small computers and use the same piece of hardware to operate them all. A variety of factors contribute to spin-offs and the faster development of embedded and IoT solutions, such as higher energy efficiency, software configuration all in one place, easier development, developed development ecosystems, and frameworks.

It should be noted that the system's ability to meet the requirements stems from the fact that the world is increasingly becoming more connected. As a result, the demand for IoT and embedded solutions will only grow. The knowledge makes all the necessary conditions exist for it to be integrated successfully by businesses and research, easing the integration of embedded devices with all types of sensors and peripherals. Researchers and businesses are at a clear advantage by using this system, as it satisfies almost all the ideal conditions for ensuring IoT serviceability, including high flexibility, modularity, and customization, alongside cloud services and containerization processes. For this reason, the number of landmark innovations changing how embedded systems impact the IoT will only increase.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Comer, D.E. and Shaughnessy, D.J. (1997) Unix and Embedded Systems. *Journal of Computer Information Systems*, **37**, 25-29.

[2] Satyanarayanan, M. (2001) Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, **8**, 10-17. https://doi.org/10.1109/98.943998

[3] Corbet, J., Rubini, A. and Kroah-Hartman, G. (2005) Linux Device Drivers: Where the Kernel Meets the Hardware. O'Reilly Media, Inc.

[4] Zaddach, J., Bruno, L., Francillon, A. and Balzarotti, D. (2014) AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares. *Proceedings*

2014 *Network and Distributed System Security Symposium*, San Diego, 23-26 February 2014, 1-16.

[5] Madakam, S., Ramaswamy, R. and Tripathi, S. (2015) Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications*, **3**, 164-173. https://doi.org/10.4236/jcc.2015.35021

[6] Samek, R. (2016) Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems. Newnes.

[7] Morabito, R., Beijar, N. and Dilworth, M. (2017) Towards Container-Based Virtualization for the Internet of Things. 2017 *IEEE International Conference on Communications* (I*CC*), Paris, 21-25 May 2017, 1-6.

[8] Franklin, G.F., Powell, J.D. and Emami-Naeini, A. (2019) Feedback Control of Dynamic Systems. 8th Edition, Pearson.

[9] Love, R. (2010) Linux Kernel Development. 3rd Edition, Addison-Wesley.

[10] Yodaiken, V. and Barabanov, M. (1997) A Real-Time Linux. *Proceedings of the Linux Applications Development and Deployment Conference* (*USELINUX-SD*97), 6-10 January 1997, California, 1-9.

[11] Yaghmour, K. (2003) Building Embedded Linux Systems. O'Reilly Media.

[12] Barr, M. (2019) Embedded Linux Primer: A Practical Real-World Approach. 3rd Edition, Prentice Hall.

[13] Pyo, C., Murao, K. and Bacivarov, I. (2019) Embedded Linux System Design and Development. CRC Press.

[14] Koziolek, H., Happe, J. and Reussner, R. (2016) Quality-Driven Software Architecture Migration. *Journal of Systems and Software*, **121**, 14-30.

[15] Sama, M.R., Rosenfeld, K., O'Connor, J. and Tuck, J. (2018) IPv6 for Embedded Systems. O'Reilly Media, Inc.

[16] Fried, L. (2005) The Linux Kernel Primer: A Top-Down Approach for x86 and PowerPC Architectures. Prentice Hall.