THE LABORATORY OF THE FUTURE: INNOVATION AND RESEARCH IN THE FIELD OF IOT

Dan Laurentiu GRECU, Lecturer. PhD, "Titu Maiorescu" University – Faculty of Informatics
Lucian Ștefăniță GRIGORE, Prof. Ph.D., Military Technical Academy "Ferdinand I"
Popa Roxan Violeta, Student at "Titu Maiorescu" University, Faculty of Informatics
https://doi.org/10.66793/titui19proceeding5

Abstract :

Innovative research laboratories play a crucial role in the development of emerging technologies, and the Internet of Things (IoT) is one of the most promising fields of study today. The creation of a laboratory dedicated to research in the IoT domain not only facilitates innovation but also contributes to the professional training of researchers and students, having a direct impact on economic and social development. This article will detail the necessary steps for establishing an innovative research laboratory in the IoT field, highlighting the importance of resources, infrastructure, collaborations, and education. An analysis will also be carried out on the main operating systems used in IoT, namely real-time operating systems – RTOS.

***Key words**: IoT, RTOS, Arduino, ESP32, Raspberry Pi, LoRa, Arduino IDE*

Introduction:

The infrastructure of the laboratories must be designed to also support research in IoT. These include hardware and software equipment, sensors, communication devices, cloud platforms, and data analysis solutions. Creating a collaborative working environment that encourages experimentation and rapid prototyping is essential.

Another important aspect is the training and attraction of qualified personnel. Implementing the laboratory in universities and higher education institutions, as well as with industry, will bring not only theoretical expertise but also practical experience. Internships and professional training programs can help young researchers become familiar with IoT technologies and develop the skills needed to meet current challenges.

Another important factor in establishing an IoT research laboratory is the integration of cybersecurity solutions. As more and more devices become connected to the Internet, protecting data and users becomes a priority. The laboratory should develop IoT solutions that safeguard both user privacy and sensitive data.

Requirements for establishing an innovative IoT laboratory

In order to physically build an innovative IoT laboratory (fig. 1), a series of essential steps should be followed for the organization of the space, the endowment with equipment, the necessary infrastructure and the technical configuration [1]. Here is a concrete detail:

1. Selection and Organization of Physical Space

• Choose a room or space where you can set up separate zones for hardware development, software testing, and collaboration.

• Provide sufficiently spacious worktables, with multiple electrical outlets and adequate lighting.

• Install modular supports and shelves for storing boards, sensors, and electronic components.

2.Acquisition of Essential Hardware Equipment

• Development boards (Arduino, ESP32, Raspberry Pi) in sufficient quantities for teams.

• Sensor and actuator kits covering as many types of measurements and control as possible.

• Measuring instruments: digital oscilloscopes, multimeters, signal generators.

• Communication equipment for building IoT networks (LoRa gateways, Wi-Fi routers).

3.Configuration of Networks and Communication Infrastructure

• Set up a separate Wi-Fi and/or LoRa network for testing devices, isolated from the main network.

• Install configurable routers and switches.

• Configure traffic monitoring and wireless spectrum tools for diagnostics.

4.Spaces and Facilities for Software Development

• Workstations equipped with IDEs for embedded programming (e.g., Arduino IDE, PlatformIO).

• Servers or computers for running virtualized environments, simulations, and managing IoT data.

• Stable internet connection for access to cloud resources and documentation.

5.Security and Protection Equipment

• Install electrical protection systems (UPS, voltage stabilizers).

• Ensure conditions for protecting sensitive electronic components (anti-static measures).

• Implement internal policies regarding equipment handling.

6.Organization of Work Processes

• Define procedures for rapid prototyping, testing, documentation, and collaboration.

• Create zones for workshops and training sessions.

• Use project management and code versioning tools (e.g., Git).

7.Collaboration and Laboratory Expansion

• Maintain connections with universities, companies, and research centers.

• Prepare space for presentations and demonstrations.

• Plan periodic equipment acquisitions and updates to stay current with IoT technologies.

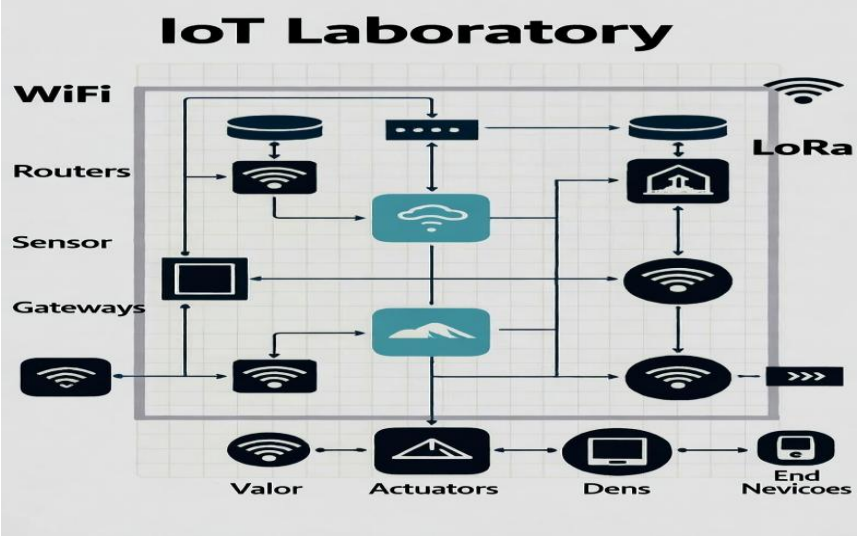

Fig. 1. Possible Hardware Architecture of an IoT Laboratory

Through these steps, it will be possible to create a functional IoT laboratory, tailored for innovative research and practical applications, providing a complete ecosystem for development, testing, and collaboration.

A fundamental aspect of an RTOS is task management. It allows the simultaneous execution of multiple tasks through the use of preemption and scheduling techniques[1][2]. Preemption is essential because it enables a higher-priority task to interrupt a lower-priority task, ensuring that the most critical processes are completed on time. Scheduling can be implemented using various algorithms, such as rate-monotonic or dynamic-priority responses, each having specific advantages and disadvantages depending on the application requirements.

Additionally, an RTOS provides facilities for managing hardware resources, such as memory and input/output. These functionalities are crucial, especially in environments with limited resources. For example, many RTOSs implement various synchronization mechanisms, such as semaphores and mutexes, which help prevent concurrency issues and ensure data integrity.

Real-Time Operating Systems (RTOS) are specifically designed for applications where strict adherence to timing constraints is essential, such as embedded systems, IoT, automotive, or aerospace. Unlike general-purpose operating systems (GPOS, such as Linux or Windows), RTOSs prioritize predictability, low latency, and resource management in hardware-constrained environments. They manage tasks through priority-based scheduling, signals, and mutexes, ensuring responses within milliseconds or microseconds..

A fundamental aspect of an RTOS is task management. It allows the simultaneous execution of multiple tasks through the use of preemption and scheduling techniques. Preemption is essential because it enables a higher-priority task to interrupt a lower-priority task, ensuring that the most critical processes are completed on time. Scheduling can be implemented using various algorithms, such as rate-monotonic or dynamic-priority responses, each having specific advantages and disadvantages depending on the application requirements.

An example of RTOS application is in the automotive domain, where engine control systems must respond quickly to variables such as acceleration, speed, and throttle conditions. These systems need to continuously monitor sensor data and adjust engine parameters in real time to ensure optimal performance and maximum safety. Other applications include drones, which require precise control of movement and stability, and medical equipment, which monitors and regulates patients' vital parameters.

Additionally, an RTOS provides facilities for managing hardware resources, such as memory and input/output. These functionalities are crucial, especially in environments with limited resources. For example, many RTOSs implement various synchronization mechanisms, such as semaphores and mutexes, which help prevent concurrency issues and ensure data integrity. Therefore, developers must be aware of efficient resource usage to avoid performance problems.

Another important aspect of RTOSs is their ability to respond quickly to events. This is crucial in critical applications, such as automotive braking systems or medical equipment, where delays can have fatal consequences. Therefore, an efficient RTOS must have a predictable and consistent response time that aligns with the application's requirements.

Although RTOSs offer numerous advantages, there are also challenges. One of these is the complexity of application development, which may require advanced knowledge in programming and system architecture (Fig. 2). Additionally, integration with specific hardware can be a complicated task, requiring rigorous testing to ensure system reliability and safety.

On the other hand, future trends in RTOSs include integration with artificial intelligence and machine learning. These technologies could allow systems to become more adaptive, learning from collected data to optimize performance. Furthermore, the development of 5G technologies will enable faster and more efficient communication, opening new opportunities for RTOSs that require constant, real-time connectivity.

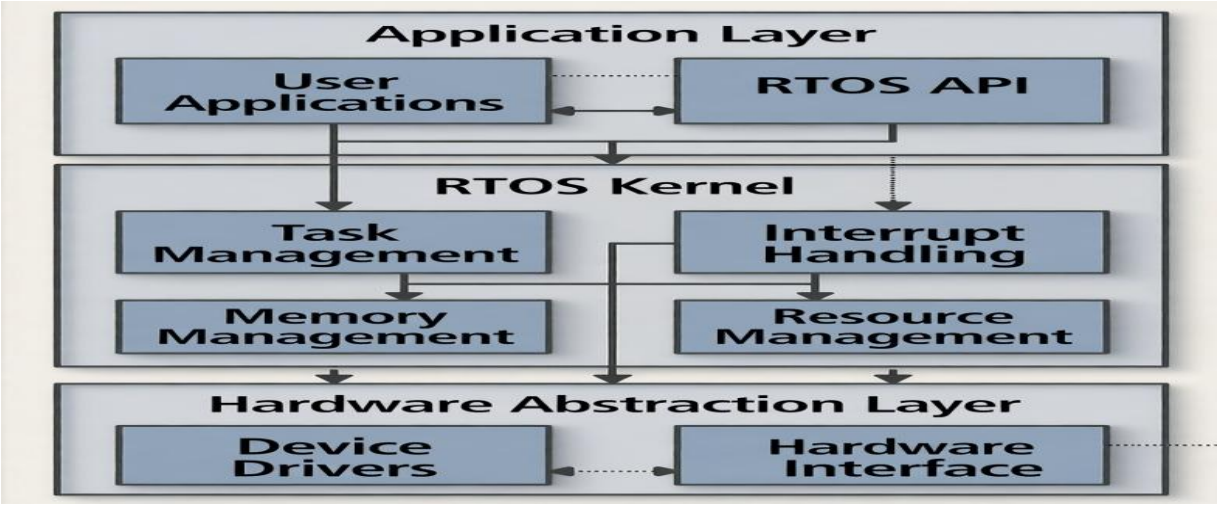

Fig. 1. Architecture of a Real Time Operating System

In this section, we detail the main RTOSs used today, based on popularity, community support, and practical applications. Six prominent RTOSs were selected for analysis: FreeRTOS, VxWorks, QNX, Zephyr, RTEMS, and µC/OS-III. These RTOSs span a wide range of requirements, from free open-source solutions to premium proprietary options[3][4].

Details on the Main Real-Time Operating Systems (RTOS)

1. **FreeRTOS**: Initially developed by Richard Barry, FreeRTOS is a lightweight, open-source RTOS oriented toward microcontrollers. It provides a minimal kernel for multitasking, with support for queues, signals, and timers. It is ideal for IoT and low-power embedded projects, offering a low learning curve. The current version includes extended support for security (MPS3) and integration with AWS.

2. **VxWorks:** Developed by Wind River Systems, VxWorks is a robust proprietary RTOS used in safety-critical applications (e.g., avionics, medical systems). It supports advanced modularity, real-time debugging, and certifications (DO-178C). VxWorks is scalable but costly, with a focus on high performance in multi-core environments.

3. **QNX:** Created by BlackBerry QNX, this RTOS is based on a modular microkernel, making it extremely stable and secure. It supports POSIX standards, networking, and POSIX-compliant file systems. QNX is widely used in automotive applications (e.g., infotainment) and industrial systems, with emphasis on fault tolerance and scalability.

4. **Zephyr:** An open-source project supported by the Linux Foundation, Zephyr is designed for IoT and resource-constrained devices. It offers a rich ecosystem of drivers, Bluetooth LE, and multi-architecture support. Zephyr is more complex than FreeRTOS but provides advanced features such as a device tree and an interactive shell.

5. **RTEMS (Real-Time Executive for Multiprocessor Systems):** An open-source RTOS developed for space and military applications (e.g., NASA). It is POSIX-compliant, supports SMP (Symmetric Multiprocessing), and is highly portable. RTEMS focuses on predictability and testability in hard real-time environments.

6. **µC/OS-III**: Developed by Jean J. Labrosse, µC/OS-III is a commercial/open-source RTOS (under Apache 2.0) with a simple API. It supports up to 65,535 tasks, profiling, and certifications (IEC 61508). It is popular in embedded microcontroller applications due to its extensive documentation.

Comparative Analysis

For a clear comparison, the data has been structured in a table based on key criteria: license (open-source vs. proprietary), supported architectures, kernel type, SMP/POSIX support, primary purpose, and advantages/disadvantages. The information is synthesized from recent sources, highlighting the trade-offs between cost, performance, and flexibility. For example, open-source RTOSs such as FreeRTOS and Zephyr are free and easily customizable but may require more effort for critical certifications, whereas VxWorks or QNX excel in high-reliability production environments.

RTOS	License	Supported Architectures	Kernel Type	SMP/POSIX	Main Embedded Sector	Advantages	Disadvantages
FreeRTOS	MIT (open-source)	ARM, AVR, RISC-V, ESP32, STM32, x86	Monolithic	No/Partial	IoT, embedded low-power	Easy to learn, large community, free	Limited to simple tasks; less mature for multi-core
VxWorks	Proprietary	ARM, PowerPC, x86, MIPS	Hybrid	Yes/Yes	Aerospace, automotive	High performance, certifications (DO-178C)	High cost; vendor dependency
QNX	Proprietary	ARM, x86-64, MIPS, PowerPC	Microkernel	Yes/Yes	Industrial, automotive	Excellent stability, modularity	Expensive license; steep learning curve
Zephyr	Apache 2.0 (open-source)	ARM Cortex, RISC-V, x86, Xtensa	Hybrid	Yes/ Partial	IoT, constrained devices	Rich ecosystem (drivers, BLE), portable	Initially more complex; higher overhead
RTEMS	BSD (open-source)	ARM, PowerPC, SPARC, x86, MIPS	Monolithic	Yes/Yes	Space, military	Predictable, full POSIX, free	Smaller community; niche focus
µC/OS-III	Apache 2.0 (open-source)	ARM, AVR, x86, PIC, STM32	Monolithic	No/Partial	Microcontrolere embedded	Simple API, scalable (many tasks)	Limited support for advanced networking

General Observations from the Comparison:

- **Performance and Predictability:** VxWorks and QNX dominate in hard real-time environments (responses <1ms) due to optimized kernels, whereas FreeRTOS and Zephyr are soft real-time, suitable for rapid prototyping
- **Cost vs. Flexibility:** Open-source RTOSs (FreeRTOS, Zephyr) reduce initial costs, while proprietary systems (VxWorks, QNX) provide enterprise support and compliance with safety standards.
- **2025 Trends**: Zephyr is gaining traction in IoT due to integration with Matter/Thread, and FreeRTOS is strengthened through Amazon's acquisition. RTEMS remains niche in research.
- **Optimal Choice:** For hobby/prototyping: FreeRTOS. For safety-critical production: VxWorks/QNX. For modern IoT applications: Zephyr.

Conclusions

In conclusion, establishing an innovative IoT research laboratory requires a comprehensive approach, including clearly defined objectives, the development of suitable infrastructure, recruitment of qualified personnel, interdisciplinary collaboration, and community engagement. Such a laboratory will not only stimulate innovation and support technological development but also contribute to training a new generation of specialists capable of addressing future challenges.

Another key conclusion is that real-time operating systems play a vital role in enabling the development of critical applications that require strict management of time and resources. Through their unique features, RTOSs allow developers to create efficient, reliable, and fast solutions, which are essential in today's technological context.

Conclusions from the Comparative Analysis of Major RTOSs [5]

1. **Diversity of Needs Dictates RTOS Choice**: There is no universal "best" RTOS; the optimal choice depends on the context. FreeRTOS and Zephyr excel in rapid prototyping and low-cost IoT applications, while VxWorks and QNX are indispensable in hard real-time safety-critical applications (aerospace, automotive, medical) due to certifications and stability.

2. **Open-Source vs. Proprietary – Balancing Cost and Support:** Open-source RTOSs (FreeRTOS, Zephyr, RTEMS, µC/OS-III) lower entry barriers and allow customization but require internal expertise for compliance and optimization. In contrast, VxWorks and QNX provide enterprise support, mature toolchains, and certifications (DO-178C, IEC 61508), justifying their higher costs in production environments.

3. **Current Trends (2025): IoT and Multi-Core Support:** Zephyr is gaining ground in the IoT ecosystem due to native support for Matter, Thread, Bluetooth LE, and device tree, becoming a modern alternative to FreeRTOS. Additionally, SMP (Symmetric Multiprocessing) is increasingly essential—VxWorks, QNX, and RTEMS offer mature support, while FreeRTOS remains limited to simple single-core systems.

4. **Predictability vs. Functionality:** RTOSs with a microkernel (e.g., QNX) provide superior module isolation and fault tolerance, making them ideal for complex systems. In contrast, monolithic kernel RTOSs (e.g., FreeRTOS, RTEMS) offer lower latency but a higher risk of error propagation.

5. **Documentation and Community – Key Adoption Factors:** FreeRTOS and µC/OS-III benefit from excellent documentation and dedicated books (e.g., Using the FreeRTOS Kernel, µC/OS-III: The Real-Time Kernel), reducing the learning curve. Zephyr compensates with the Linux Foundation ecosystem but remains initially more complex.

6. **Strategic Recommendation:**
    - **Prototyping / Hobby / Education: FreeRTOS**
    - **Connected, scalable IoT: Zephyr**
    - **Certified, safety-critical systems: VxWorks or QNX**
    - **Space / Military research: RTEMS**
    - **Simple, high-performance microcontrollers: µC/OS-III**

In conclusion, the choice of an RTOS must align with timing constraints, hardware, budget, and certification requirements. The rapid evolution of Zephyr and the integration of FreeRTOS with cloud services (AWS) indicate a future convergence between embedded and edge computing.

References

1. STAR-UBB-N. (2023). *Virtual Labs (VL) – direcții de cercetare*. Universitatea Babeș-Bolyai, Cluj-Napoca. Disponibil online: https://cercetare.ubbcluj.ro/wp-content/uploads/2023/11/RO_competitie_fellows_PFE-550-UBB-etapa-V-27.11.2023_Anexa-1.pdf
2. Baskiyar, S., & Kalé, V. (2001). *A Survey of Contemporary Real-time Operating Systems*. University of Cincinnati. Disponibil la: http://robertdick.org/esds/papers/baskiyar-rtos.pdf.
3. Aroca, R. V., & Caurin, G. A. P. (2011). *A Real Time Operating Systems (RTOS) Comparison*. Journal of Physics: Conference Series, 289(1). Disponibil la: https://www.semanticscholar.org/paper/A-Real-Time-Operating-Systems-%28-RTOS-%29-Comparison-Aroca-Caurin/3dc51976f8fb9408f5c991d457fa27f7b5adb737.
4. Neuhäußer, L., et al. (2022). *A Comparison of Real-time Operating Systems for Embedded Systems*. ResearchGate. Disponibil la: https://es.cs.rptu.de/publications/datarsg/Neuh22.pdf.
5. Kalpen, S. (2001). *Performance Comparison of RTOS*. Columbia University. Disponibil la: https://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/reports/shamil-kalpen.pdf.