RESEARCH ARTICLE OPEN ACCESS

Real-Time Operating Systems for Safety-Critical Applications (e.g., Drones, Medical Devices)

Dr.Mohammed Iliyas
Faculty, Dept. Of Computer Science,
Adikavi Sri Maharshi Valmiki University, Yergera Road, RAICHUR,
Karnataka, INDIA,

ABSTRACT
Real-Time Operating Systems (RTOS) play a vital role in safety-critical applications where timely and predictable responses are essential, such as in drones, autonomous vehicles, medical devices, and industrial automation systems. These applications demand stringent timing constraints, reliability, and fault tolerance to ensure human safety and system stability. This paper explores the architectural principles, scheduling techniques, and communication models that underpin RTOS functionality in safety-critical environments. Key features such as deterministic behaviour, priority-based primitive scheduling, and inter-process communication mechanisms are analysed in depth. Additionally, we investigate the integration of safety standards like ISO 26262, DO-178C, and IEC 62304 within RTOS frameworks. The paper also highlights recent advancements in lightweight RTOS for resource-constrained embedded systems and examines their implementation in real-world platforms such as FreeRTOS, VxWorks, and QNX. Case studies on drone navigation and patient monitoring systems are presented to demonstrate practical use cases. Major challenges such as security vulnerabilities, mixed-criticality task management, and certification complexities are discussed. Finally, we propose future research directions involving AI-enabled real-time systems, formal verification, and cyber-physical system resilience. This study aims to serve as a foundational reference for researchers and developers seeking to enhance the reliability and intelligence of RTOS in safety-critical domains.

Keywords — Real-Time Operating Systems (RTOS), Safety-Critical Systems, Deterministic Scheduling, Embedded Systems, Fault Tolerance and Certification.

I. INTRODUCTION

In today's technology-driven world, real-time computing has become essential in numerous domains where system correctness depends not only on logical results but also on the time at which results are produced. Safety-critical applications—such as those found in autonomous drones, automotive control systems, medical implants, industrial automation, and aerospace systems—demand extremely reliable and time-predictable computing environments. The cornerstone of such systems is the Real-Time Operating System (RTOS), which provides deterministic task scheduling, low-latency interrupt handling, and predictable execution behaviour that traditional general-purpose operating systems cannot guarantee. An RTOS is specifically engineered to manage hardware resources, run tasks, and respond to external inputs within strict time constraints. Unlike conventional operating systems, where performance is often optimized for throughput or resource utilization, RTOS are designed for predictability, ensuring that critical tasks meet their deadlines consistently. In safety-critical systems, failure to meet these deadlines can lead to catastrophic consequences such as loss of human life, environmental damage, or economic loss. The proliferation of embedded systems and cyber-physical systems (CPS) has further emphasized the need for robust real-time operating systems. For instance, in unmanned aerial vehicles (UAVs), RTOS governs sensor fusion, flight control, and obstacle avoidance—each with specific timing requirements. Similarly, in the healthcare industry, medical devices like pacemakers and infusion pumps rely on real-time systems to ensure precise operation and timely intervention. In such applications, the RTOS must not only schedule and manage tasks efficiently but also provide fault tolerance, real-time inter-process communication, and compliance with safety certifications like DO-178C, ISO 26262, and IEC 62304.

The design of RTOS for safety-critical applications faces multiple challenges. These include handling mixed-criticality workloads, ensuring temporal isolation, dealing with resource constraints in embedded environments, and maintaining system certification across frequent updates. The evolution of computing architectures—multi-core processors, heterogeneous computing units, and AI accelerators—has added complexity to real-time system design, demanding novel scheduling algorithms, memory models, and synchronization mechanisms. Several RTOS platforms such as VxWorks, QNX Neutrino, FreeRTOS, RTEMS, and Zephyr have gained prominence in different industrial sectors. Each comes with a trade-off between performance, scalability, certification readiness, and community support. As real-time systems continue to evolve, there is a growing interest in integrating machine learning for predictive scheduling, leveraging formal verification for provable safety, and embedding security mechanisms at the OS level to protect against cyber-attacks—another major concern in connected safety-critical systems.

This paper presents a comprehensive study of real-time operating systems in safety-critical domains. It reviews their architectural foundations, examines current challenges in design and deployment, and discusses the integration of standards and security requirements. Case studies in aerospace and healthcare domains illustrate real-world implementations. Finally, the paper outlines future directions such as AI-enabled real-time management, hybrid scheduling, and quantum-aware real-time systems. By providing a holistic view, this work aims to contribute to the advancement of secure, reliable, and intelligent RTOS for the next generation of safety-critical applications.

II.OBJECTIVES:

- To explore the fundamental architecture and characteristics of Real-Time Operating Systems (RTOS).
- Understand how RTOS differ from general-purpose operating systems in terms of scheduling, interrupt handling, and task management.
- To analyse the role of RTOS in safety-critical applications such as drones, autonomous systems, and medical devices.
- Demonstrate how RTOS ensures deterministic behaviour, reliability, and timing precision in real-world use cases.
- To examine the key safety, fault tolerance, and security measures integrated into RTOS for mission-critical environments
- Highlight techniques such as redundancy, watchdog timers, memory protection, and secure boot mechanisms.
- To review and compare commonly used RTOS platforms (e.g., FreeRTOS, QNX, VxWorks) in safety-critical systems.
- Evaluate their capabilities, certification readiness, and suitability for different application domains.
- To identify and discuss the challenges associated with RTOS in safety-critical systems
- Address issues like mixed-criticality task handling, certification complexity, and vulnerability to cyberattacks.
- To align RTOS design with global safety and software standards (e.g., ISO 26262, DO-178C, IEC 62304)
- Discuss how these standards influence the development lifecycle and compliance strategies.
- To propose future research directions and innovations in RTOS for enhanced safety and intelligence
- Explore trends such as AI-based scheduling, formal verification, and adaptive real-time systems.

These objectives aim to provide a comprehensive understanding of how RTOS are designed, implemented, and evolved to meet the stringent demands of safety-critical applications.

IV.SAFETY MEASURES.

Here are some key safety measures that must be implemented in safety-critical applications such as drones, medical devices, and similar systems built on Real Time Operating Systems(RTOS)

1. Deterministic Task Execution:

Purpose: Ensures critical tasks meet deadlines without unpredictable delays.

Measures:

Use of priority-based pre-emptive scheduling.

Worst-case execution time (WCET) analysis for all real-time tasks.

Time-triggered architectures for periodic safety checks.

2. Fault Tolerance and Redundancy:

Purpose: Prevent system failure during faults or hardware errors.

Measures:

Watchdog timers to reset the system in case of software hang.

Redundant components (e.g., dual sensors or processors).

Graceful degradation to a safe state during faults.

3. Security Enforcement:

Purpose: Prevent malicious attacks that could compromise safety.

Measures:

Secure boot and code signing to prevent unauthorized firmware.

Memory protection units (MPUs) for isolating processes.

Data encryption and secure communication protocols (e.g., TLS, DTLS).

4. Compliance with Safety Standards:

Purpose: Ensure systems meet certified industry regulations.

Standards:

ISO 26262 – Automotive Functional Safety.

DO-178C – Software for airborne systems.

IEC 62304 – Medical device software safety.

5. Real-Time Health Monitoring and Logging:

Purpose: Continuously monitor system state and allow post-event analysis.

Measures:

Runtime diagnostics and health checks.

Event logging and trace recording for audit and failure analysis.

Self-check routines for hardware and software components.

6. Formal Verification and Testing:

Purpose: Validate that the system behaves correctly under all scenarios.

Techniques:

Model checking and formal verification of scheduling and communication logic.

Unit, integration, and system-level testing with hardware-in-the-loop (HIL) simulation.

Stress and boundary condition testing.

7. Fail-Safe and Safe-State Design:

Purpose: Ensure the system enters a safe state in case of failure.
Examples:
Drone automatically lands if GPS or IMU fails.
Medical device switches to manual control mode on software fault.
Power cut-off mechanisms in case of thermal overload or malfunction.

8. Controlled and Secure Update Mechanisms:
Purpose: Prevent system corruption during software updates.
Measures:
Use of OTA (Over-The-Air) update protocols with rollback options.
Updates must be cryptographically signed and verified before installation.

III.BRIEF EXPLAINATIONS.

Real-Time Operating Systems (RTOS) are crucial in safety-critical applications where the correctness of system behaviour is determined not only by the logical results but also by the timing of those results. In systems such as drones, autonomous vehicles, medical devices, and industrial controllers, a delay or missed deadline can lead to catastrophic outcomes including system failure, injury, or even loss of life. RTOS are specifically designed to meet strict timing, predictability, and reliability requirements through features like deterministic scheduling, low interrupt latency, and real-time task management. In drones, for instance, RTOS ensures that flight control loops, sensor data processing, and obstacle avoidance routines execute within their defined time windows. A missed deadline in flight stabilization could lead to an immediate crash. Similarly, in medical devices like pacemakers or infusion pumps, timing accuracy is essential to deliver therapy at the right intervals. RTOS guarantees that critical medical processes execute precisely and reliably, even under varying system loads. Key characteristics of RTOS in these applications include pre-emptive multitasking, inter-task communication, priority-based scheduling, and memory protection. Furthermore, RTOS used in safety-critical systems often need to comply with international safety standards such as ISO 26262 (automotive), DO-178C (aerospace), or IEC 62304 (medical devices). These standards define rigorous guidelines for system design, testing, verification, and certification. Another important aspect is fault tolerance. RTOS must include mechanisms like watchdog timers, redundancy, and safe-state fullback strategies to handle system errors gracefully. Security is also a growing concern, as many safety-critical systems are now connected via networks or the cloud, making them vulnerable to cyber threats. RTOS provide the foundation for building reliable, secure, and time-deterministic safety-critical systems that protect lives and assets across various domains.

V.DISCUSSION AND CONCLUSION:

Real-Time Operating Systems (RTOS) serve as the backbone of safety-critical applications where timing precision, reliability, and fault tolerance are non-negotiable. In domains such as drones, autonomous systems, and medical devices, RTOS ensure that vital tasks are executed within strict time constraints, thereby maintaining system integrity and safeguarding human lives. Through features like deterministic scheduling, low-latency interrupt handling, and robust inter-process communication, RTOS provide the necessary framework for developing dependable real-time systems.
This paper has explored the architectural foundations, safety mechanisms, and compliance requirements that underpin RTOS in critical environments. It has also examined key challenges such as mixed-criticality task management, security vulnerabilities, and the complexity of regulatory certification. With examples from widely-used platforms like FreeRTOS, VxWorks, and QNX, we illustrated how RTOS are implemented in practical, high-stakes scenarios. As embedded and cyber-physical systems become increasingly complex and interconnected, the need for more intelligent, secure, and adaptive real-time systems continues to grow. Future research should focus on integrating machine learning for predictive scheduling, enhancing system verification through formal methods, and strengthening cyber security at the OS level. Ultimately, RTOS will remain central to the advancement of safe, efficient, and intelligent technologies in critical domains.

VI. REFERENCES:

[1].**Yew Ho Hee et al.**, “Innovations in Real-Time Operating Systems (RTOS) for Safety-Critical Embedded Systems” Reviews ARINC 653-style partitioned RTOS architecture (e.g., VxWorks 653, PikeOS), formal verification techniques, and autonomous vehicle use-cases.
[2].**Parkinson & Gasperoni**, “High Integrity Systems Development for Integrated Modular Avionics Using VxWorks and GNAT” (2002) Classic IMAvionics case study on DO-178C processes with VxWorks—cited in broader AVIONICS safety research.
[3]**Wind River**, *VxWorks Cert Edition Product Overview* (2025) Details DO-178C DAL-A, IEC 61508 SIL 3, ISO 26262 ASIL-D, IEC 62304 Class C compliance in modern VxWorks RTOS reddit.com+13windriver.com+13
[4]**QNX OS for Safety Datasheet**, BlackBerry (2020–2025) Covers microkernel architecture, ISO 26262, IEC 61508, IEC 62304 certification, fault isolation, and certification chain-of-trust
[5].**Steve Bush**, “PX5 RTOS with Arm TrustZone Support” (2023) Introduces a lightweight, resource-efficient RTOS supporting MISRA compliance and POSIX threading—useful for constrained embedded environments
[6]. **Anand Eswaran et al.**, “Nano-RK: an Energy-aware Resource-centric RTOS for Sensor Networks” (2005) Describes priority-based scheduling and inter-process communication in deeply resource-constrained systems

[7].**Yongwang Zhao et al.**, "Event-based Formalization of Safety-critical Operating System Standards: An Experience Report on ARINC 653 using Event-B" (2015) Formal verification of RTOS scheduling and partitioning standards, uncovering semantic errors in ARINC 653.

[8].**Ankush Desai et al.**, "SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems" (2018) Demonstrates a drone navigation case study using runtime safety wrappers and controller switching—merging formal methods with real-time control .