Real-Time Operating Systems (RTOS): A Survey on Use Cases and Challenges

Sukhada Harsulkar, Rutuja Darade, Riya Dargude, Nayan Loya, Minal Deshmukh.
Department of Electronics and Telecommunication Engineering
Vishwakarma Institute of Information Technology Pune, India
sukhada.22210857@viit.ac.in
minal.deshmukh@viit.ac.in
rutuja.22210021@viit.ac.in
riya.22211221@viit.ac.in
nayankumar.22211594@viit.ac.in

Abstract— [1] Real-time operating systems (RTOS) are the fundamental of today's embedded systems that provide precise control and time-sensitive characteristics in the field of different applications. This paper explicates the various uses of RTOS such as using TinyOS in wireless sensor networks for gathering smart data in an energy efficient way and the use of tasking methods for transmitting data safely and efficiently. Smart assistant devices for the visually impaired present an example of RTOS in combination with malleable human-interface technologies, while RTOS-supported robotics systems demonstrate the progress made in realtime kinematic control and multithreaded implementation. Other introductions are: RTOS Extracter for performing automated system analysis; protected memory domain concepts aiming at achieving fault isolation; and/or application-targeted RTOS architectures addressing strict safety sensitivity levels. The paper also discusses dynamic scheduling for the event-driven industrial automation system and the actual IEC 60870-5-103 standards for industrial control systems communication, providing accurate compatible integration.

Tackling major issues, this work discusses the best software architecture for cyber-physical systems and round-robin CPU scheduling with dynamic time quantum modification for better throughput and latency improvement. This multi-faceted problem involves guaranteeing control flow integrity against runtime threats and developing fine-grained lightweight monitoring mechanisms for performance enhancement as central tasks to increase RTOS dependability.

An application of real-time GPS with gyro compassing employing cascaded EKF is coupled with the FreeRTOS represents worth mentioning fusion of location and orientation. Moreover, RT-ATDE.

Keywords: Operating systems, especially Real-Time Operating Systems (RTOS), embedded systems, Tiny Real-Time Operating Systems, sensor networks, task-based techniques, conjugate adaptive human-computer interface, RTOS based robotic systems, dynamic scheduling techniques, security-sensitive systems, IEC 60870-5-103, memory protection schemes, cyber-physical systems, control flow integrity, performance measurement, Extended Kalman Filters (EKFs), FreeRTOS, GPS real-time integration.

I. INTRODUCTION

RTOS is essentially in the core of embedded systems as they supply successful temporal scheduling for great performance rate as per the taste of present days application. As the features of embedded technologies advance, there had been a growing need for RTOSs that can deliver high-performance, low-latency and safety-critical applications. RTOS facilitates determinism by such attributes as, preemptive multitasking, priority scheduling, and real-time clock control making it mandatory where industrial automation, robotics, healthcare, and IoT operations are involved.

In this paper, the broad functionality of RTOS in different advanced use is discussed. As an example, TinyOS, a lightweight event driven RTOS used extensively in WSNs, has shown enhanced energy management and security for data transmission. These capabilities are expanded on by

task-based approach which fosters scalable and more robust communication frameworks. RTOS also continues to have a revolutionary role in human oriented technologies including smart helping gadgets to the blind, using real time adaptability for user interface interaction. Dynamic scheduling increases capability in event-responsive applications in industrial environments more so DyROS and the robotic systems based on RTOS demonstrate improvements in multi-task control and sensors. Additionally, the RTOS frameworks designed for the safety-critical applications include its importance to aerospace, automotive, and medical systems that guarantee the feature's resilience and functionality.

However, RTOS is not immune to some major challenges that require new approaches for their solution. These are the software design for cyber-physical systems, improved CPU scheduling algorithms with flexible time quantum and proper control flow to avoid runtime problems. Subtle performance monitoring procedures include instrumentation that enables the acquisition of more refined details concerning the efficiency of RTOS, for effective performance assessment and tuning. Further, spline interpolation with Kalman filters such as GPS navigation and gyro compassing using cascaded EKFS with FreeRTOS show how complicated real time systems are and how significant the problem of optimal scheduling is. Last but not least, practical tools such as RT-ATDE (Real-Time Analysis and Testing Development Environment) can be viewed as the means of transitioning from formal models and bringing real-time constraints into validation [1].

II. BACKGROUND

A.3.1 FRAMEWORK: [2] The system of an RTOS serves as the foundational engineering that empowers the execution of planning and assignments inside strict timing imperatives. Broadly two essential models characterize the basic organizations of RTOS systems; library-based and isolated executable models. The centre functionalities of RTOS spin around assignment planning, memory administrations and inter-task communication. Real-time planning approaches like rate-monotonic or earliest-deadline-first are commonly utilized to meet strict timing prerequisites. Memory administration regularly incorporates equipment highlights Memory Security Units (MPUs) to accomplish special confinement, which is basic in safety-critical spaces. Furthermore, inter-task communication instruments such as message lines and semaphores play a vital part in guaranteeing compelling assignment coordination.

In safety-critical applications, RTOS must comply with thorough industry benchmarks, such as IEC-61508 or DO-178C. These systems coordinated security highlights like blame resilience and comprehensive confirmation forms to meet the rigid necessities of spaces like aviation, car and mechanical robotization. By tending to these challenges, the RTOS gives a strong establishment for real time operations, guaranteeing unwavering quality and effectiveness in basic frameworks.

B.3.2 ARCHITECTURE: The engineering of an RTOS is planned to guarantee the convenient execution of errands whereas overseeing framework assets proficiently. It is ordinarily organized around key components such as the bit, errand administration, memory administration, and inter-task communication mechanisms. The bit serves as the centre of the RTOS, overseeing assignment planning and asset assignment. It underpins different planning calculations, counting fixed-priority and energetic need plans, such as Rate-Monotonic Planning (RMS) or Most punctual Due date to begin with (EDF). These planning approaches are basic in ensuring that time-sensitive assignments meet their due dates beneath exacting constraints. In RTOS-based frameworks, errands are frequently organized as autonomous program units with clearly characterized execution necessities. The engineering bolsters synchronous and offbeat communication between these errands, utilizing signalling instruments like semaphores, message lines, and shared memory. These highlights empower dependable coordination in complex frameworks, such as mechanical mechanization or therapeutic devices. The plan of RTOS models moreover consolidates memory administration methodologies custom fitted for the system's equipment imperatives. Numerous RTOS stages utilize memory security units (MPUs) to accomplish spatial confinement between assignments. This approach not as it were upgrades security by avoiding memory debasement but too encourages secluded framework improvement by permitting errands to work freely inside their distributed memory spaces [2].

C. 3.3 ROLE OF THE RTOS: In real-time frameworks, the working framework (OS) plays a basic part in guaranteeing the deterministic execution of assignments whereas overseeing framework assets successfully. The principal objective of a real-time working framework (RTOS) is to meet the timing and execution prerequisites of errands inside particular limitations, which is fundamental for the

usefulness and unwavering quality of implanted and safety-critical applications.

The RTOS part acts as the central chief, capable for planning errands concurring to real-time needs and guaranteeing that high-priority errands are executed without undue delay. To accomplish this, RTOS employments pre-emptive or agreeable multitasking plans, frequently complemented by calculations like Rate Monotonic Planning (RMS) or Most punctual Due date to begin with (EDF), which ensure schedulable beneath particular conditions.

Memory administration in an RTOS guarantees spatial segregation between errands to avoid memory debasement and empower blame control. This is especially critical in safety-critical applications where assignments of changing criticality levels may coexist. Highlights such as memory assurance units (MPUs) or memory administration units (MMUs) are utilized to implement boundaries and improve framework security.

III. OPTIMIZING RTOSPERFORMANCE

A. Resource Management: Resource Management in RTOS is a basic viewpoint of guaranteeing the proficient and solid operation of real-time frameworks. RTOS utilize synchronization components like mutexes and semaphores to oversee asset sharing among errands. This is basic in avoiding information debasement and race conditions, which can compromise the keenness of the system. In RTOS, asset administration is regularly taken care of by the part, which is mindful for overseeing framework assets and giving administrations to applications. The bit incorporates a scheduler, memory administration, hinder taking care of, and assignment administration components, all of which play a significant part in asset management. [2]

B. Memory Management in RTOS:
Fixed Partitioning: Fixed Partitioning is a memory administration strategy utilized in RTOS where the accessible memory is partitioned into fixed-size pieces. Each square is distributed to a particular errand or handle, and the estimate of the square is decided at compile time. This procedure guarantees proficient memory allotment and deallocation, as the measure of the memory piece is settled and known in progress. The points of interest of Fixed Partitioning incorporate effective memory assignment and deallocation, unsurprising memory utilization, and moo overhead. Be that as it may, the drawbacks incorporate unbendable memory allotment and wastage of memory if the designated piece is bigger than the required size. [2]

1) Dynamic partitioning: Dynamic Partitioning is a memory administration procedure utilized in RTOS where the accessible memory is distributed powerfully, based on the prerequisites of the framework. The estimate of the memory piece is decided at runtime, and the memory is apportioned and deallocated as required. The preferences of dynamic partitioning incorporate adaptable memory allotment and proficient utilize of memory. Be that as it may, the drawbacks incorporate higher overhead due to energetic assignment and deallocation, and potential for memory fracture. [2]
2) Memory pooling: Memory Pooling is a memory administration method utilized in Real-Time Working Frameworks (RTOS) where a pool of accessible memory is kept up, and when an errand or prepare requires memory, it is designated from the pool. This strategy diminishes memory fracture and progresses framework execution by minimizing the overhead of energetic memory allotment and deallocation. The points of interest of Memory Pooling incorporate decreased memory fracture, progressed framework execution, and proficient utilize of memory. Be that as it may, the drawbacks incorporate higher overhead due to pool administration, potential for memory fatigue if the pool is not overseen legitimately, and the require for cautious pool measure estimation to dodge memory squander or weariness. [2]

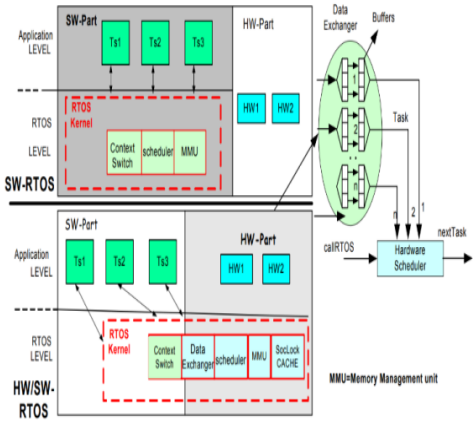

Figure 2: Software Real-Time Operating System (SW-RTOS) versus HW/SW-RTOS

Fig.2. Software Real Time Operating System (HW RTOS verses SW RTOS). [3]

IV. CASE STUDY AND IMPLEMENTATION

4.1. This case study looks into a method through which open-source Real-Time Operating Systems (RTOS) may be transformed into application specific solutions appropriate for environments where safety is paramount such as urban air mobility and avionics. Safety-critical systems should adhere to stringent requirements such as DO-178C in aviation to ensure that even if faults occur, the resulting effects are not catastrophic. The paper elaborates on the challenges that non-commercial RTOS, use, are disadvantageous as they often lack the requisite certification artefacts and formalized development processes for high assurance environments. The methods employed in the study included assessment and customization of RTOS candidates in line with some safety critical application principles, such as static allocation of memory, coding standards MISRA-C, and dependable fault management. The authors conducted a particular study in the context of an airline onboard maintenance system which has to comply with DO-178C Development Assurance Level D (DAL-D) and demonstrate the approach. The emphasis is first put on exploring various open-source RTOS alternatives while scrutinizing their licensing and configuration features. After that, non-negotiable constraints such as static resource allocation and the complete suppression of conditional compiling directives which can impact kernel security are implemented on a chosen RTOS. Finally, the modified RTOS and its certification artifacts are used in the integration process.

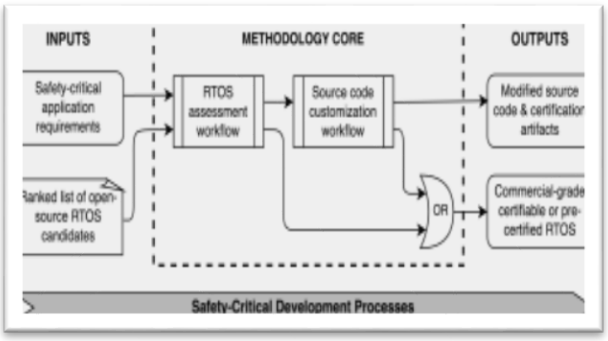

Fig.2. An overview of the application-specific RTOS for safety-critical systems methodology

RTOS Implementation: This is the case for implementing a Real-Time Operating System as part of the safety-critical development process; strict rules and procedures must be adopted in order to guarantee the performance, safety, and dependability of the system. RTOSs should support real-time operations with consistent timing behavior and strong fault-tolerant mechanisms especially in safety-critical applications such as industrial control, automotive, aerospace, and medical equipment. A detailed requirements analysis is normally the initial step of development in which the RTOS's performance and safety requirements are defined. This is followed by selecting or designing an RTOS that adheres to applicable industry standards like ISO 26262, DO-178C, or IEC 61508. The RTOS then becomes part of the system architecture such that it supports all the essential safety-critical functionality including priority-based preemption, deterministic scheduling support, and support of inter-process communications. For reducing the failure likelihood, safety features like memory protection, error management, and system health monitoring are integrated. Techniques applied for testing include static analysis, dynamic analysis, and formal verification, to mention just a few. The real-time restrictions are thoroughly checked to ensure that the system satisfies deadlines in all operational scenarios. Another step in the process is creating full documentation to back certification attempts, which may include audits from regulatory bodies. RTOS updates and maintenance should be performed with care, ensuring backward compatibility and revalidation to preserve compliance. Given that the RTOS implementation follows a systematic and standards-compliant methodology, it guarantees dependability and security required for mission-critical applications in safety-critical systems.[4]

4.2 In this paper, one such use case has been described—implementing a memory protection technique within a tailored Real-Time Operating System intended for safety-critical applications—the particular example here is aimed at resource-constrained devices, as is the case in ARM Cortex-M microcontrollers. The method promised here satisfies the requirements of safety-critical because using a Memory Protection Unit allows spatial isolation, preventing defects inside one job or application from interfering with others. In this use case, the RTOS runs on an ARMv7-M member, the STM32H7 MCU. The system is configured based on an XML approach where application and task definitions exist as well as their associated resources, such as interprocess communication facilities and memory regions. Tasks within the same application share a common memory space and can use shared-memory communications. Communication between tasks in different applications will be strictly enforced through

message passing for isolation. To begin with, the existing memory areas are defined statically in advance using examination of aperformed applications, to avoid object code conflicting with position independent module and hardware restrictions. The configuration of the MPU is handled by the RTOS kernel when switching tasks, maintaining a level of protection over transformation activities. This configuration takes into account memory regions containing the code, global variables and the opening stack as well as other aspects including cache policy and access control for memory mapped peripherals. The use case also demonstrates the system's flexibility through examples covering tasks related to an IPERF network performance server, network data handling, and associated memory access and buffering regions. Such an approach, by dealing with memory alignment, memory optimization and hardware limitations, achieves memory efficiency and enhanced fault containment with the safety critical applications running on a limited hardware resources. [5]

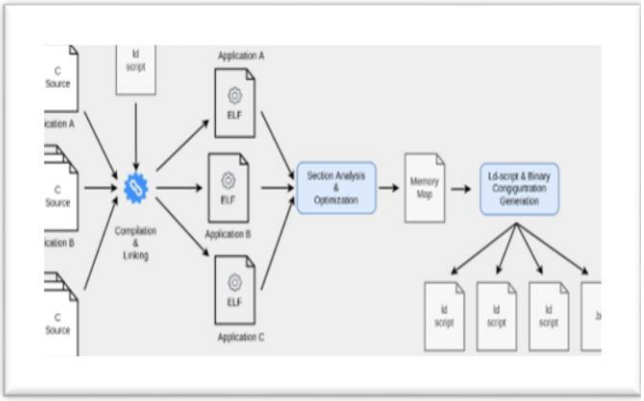

Fig.3 refered from paper Workflow for defining protected memory-regions.[5]

RTOS Implementation: The paper provided explores the use of an RTOS in a safety-critical environment with an emphasis on memory protection and spatial isolation policies, even when working with highly resource constrained systems that only have rudimentary Memory Protection Units (MPUs). The design and architecture of the RTOS is aimed at real-time systems that are involved in sensing and actuating processes, which is mostly in automotive, aerospace and medical industries. Since these microcontroller units (MCUs) do not usually come with advanced MMUs, the thrust of the design depends on the usage of the MPU to achieve spatial isolation of the executing threads. This method divides the responsibilities into different programs, and each program has its own independent memory address space. Sharing of data within the same program is done via shared memory. However, there is no such provision for communications between applications and such communications are always message based. The RTOS kernel manages the MPU during operation in a way that none of the tasks works on any other memory address space apart from the designated ones, which eliminates application crashing effects. In a similar way, access to memory-mapped devices is also restricted to the applications that are allowed to use that device. The implementation procedure commences with the assessment of the memory requirements of each individual application, which consists of code, data, stacks, and other sections. In this route a memory map is created to improve the memory within the hardware limits which also considers the address alignment. Memory protection regions are specified at this stage which is also utilized by the RTOS kernel during execution. The system achieves post-production memory management adherence to the design constraints of safety-critical systems. One of the critical aspects of the strategy is that it is flexible and can be integrated into different MCUs without extensive modifications. It also allows for resource configurability making it appropriate for use in applications that depend on pre-certified elements. In an industrial case study involving ARMv7-M based MCUs the strategy was implemented successfully proving its efficiency in utilizing the memory and providing safety in a constrained environment. [5]

Assessment	Component	Measure
Functional	*Product*	Functionality
	Environment	Interoperability
		Legal
Performance	*Product*	Usability
		Efficiency
		Portability
		Correctness
		Reliability
		Certifiability
	Environment	Software Design Process
		Business
		Product History
Non-functional	*Product*	Life Cycle
	Environment	Cost

Fig.4. Measure Characterization. [5]

4.3. The system mentioned in the above reference, Internet of Farming Things and RTOS-based Robotic System for Water Quality Monitoring and Fish Feeding in Freshwater Aquaculture, is a strong example of how the practical application of Real-Time Operating Systems makes them answer numerous complex challenges in aquaculture. The use case here discusses incorporating RTOS with IoT to

monitor and manage water quality as well as fish-feeding processes efficiently. In the proposed system, RTOS plays a key role in coordinating various tasks of real-time data acquisition, processing, and actuation in freshwater aquaculture. The system comprises a robotic module equipped with several sensors for monitoring key water quality parameters, such as temperature, pH, turbidity, and DO. These parameters are analyzed using a Water Quality Suitability Index to ensure optimal conditions for fish health and growth. This enables the RTOS to perform these tasks in a round-robin manner and, therefore, ensures that operations like monitoring and feeding are accurately and promptly carried out. It operates independently, floating on the top of water and covering pre-set areas and gathering data, distributing feed and entails a cloud connected stage for storage and visualization of all data with the capability to monitor and regulate in real-time, using a mobile application in a system that enables it for farmers to access critical information remotely and make any feeding time or quantity adjustment, feeding time, and quantity based on real-time insights.

Key features include:

Sensor Integration and Control: RTOS ensures real-time data collection from sensors measuring water quality parameters, plus executes control algorithms for feeding mechanisms.

Real-Time Feedback and Adaptation: The system dynamically adjusts feeding schedules based on water quality data and the behavior of the fish, thereby optimizing feed utilization.

Energy Efficiency: A solar-powered module ensures sustainable operation, with RTOS managing power distribution and resource optimization.

Cloud-based Mobile App Integration: Data is processed and stored in the cloud, enabling visualization and remote control through user-friendly applications. This system is an example of RTOS capability in efficiently handling concurrent processes with high reliability and low latency. However, it also presents some limitations, such as to ensure robust connectivity in rural areas, calibration of sensors, and handling possible hardware failures. The significant limitation is reliance on reliable internet availability for real-time monitoring and control, which would be best addressed by the integration of edge computing functionalities in future versions. Finally, the IoT-enabled RTOS-based robotic system culminates all the real-time decision-making, efficiency improvement, and automation potential of the RTOS in aquaculture. This use case would therefore align perfectly with your survey paper on the broader theme by illustrating the practical application and challenges of deploying RTOS in dynamic environments.[6]

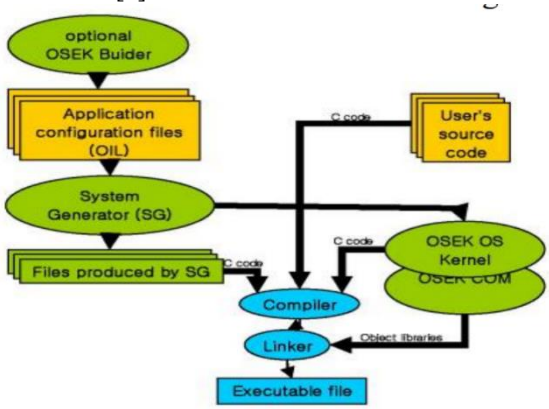

Fig.5.Example of development process for applications. [6].

Performance evaluation: Real-time measurements of task switching time, ISR handling time, and alarm service time were carried out to evaluate the OSEK turbo system performance. A worst-case execution time (WCET) approach is studied in this paper, as it is crucial for ensuring timely execution of tasks in stringent real-time conditions, examples of which can be automotive systems. Findings: Among other things, the study measured entry latency for interrupts, the time required to activate a task, the timing for switching from ISR to the task, and the time taken to activate a task upon alarm arrival. These parameters presented help in the computation of the WCET and overall performance of the system. The results confirmed that the system is capable of multitasking and meeting the time constraints of completing high priority tasks. The conclusion of this work accentuates the relevance of employing RTOSs like OSEK for automotive ECUs which require real-time task management. The study also demonstrates how OSEK turbo provides a way of managing tasks efficiently in complex automotive systems through the analysis of task switching times and other performance metrics. [6]

4.4. The paper, "Dynamic Scheduling for Event-Driven Embedded Industrial Applications," focuses on the challenges and advancements in optimizing real-time operating systems (RTOS) for embedded industrial applications with event-driven task models. This work introduces a novel lightweight dynamic scheduler integrated into FreeRTOS to improve schedulability while reducing the overhead in timing and memory. This paper provides an integration of a dynamic scheduling policy suited for the nature of event-driven tasks with industrial scenarios. Unlike regular periodic task scheduling, event-

driven tasks are triggered in unpredictable external or internal events. The authors implemented a dynamic scheduler, specifically using the earliest deadline first policy within FreeRTOS. Improved performance has been demonstrated in such scenarios as smart home devices, such as Nespresso coffee machines.

Key improvements include:

Reduced Deadline Misses: Event-driven dynamic scheduling improved missed deadlines by up to 60% as compared with static scheduling.

Efficiency Improvements: The dynamic scheduler reduced the average overhead of timing due to task insertion and selection by 34.7% and reduced memory overhead by up to 74.7%.

Case study: Real-world case of a coffee machine application that has computationally intensive tasks, such as capsule recognition using AI. Dynamic scheduling can help balance high-frequency and long-duration tasks. Relevance to RTOs and Real-Time Applications:

This work underlines the necessity of event-driven task models for RTOS, since traditional periodic scheduling cannot take into consideration the changing demands of modern cyber-physical systems. The integration of this lightweight dynamic scheduler into FreeRTOS shows that such enhancements can be achieved with minimal resource overhead; it will be feasible in constrained industrial environments. It gives a major leap in the handling of real-time constraints while supporting advanced features like embedded intelligence. [7]

Assessment and Testing: An extensive series of performance tests were conducted on the modified RTOS to evaluate its quality of service and performance. Latency Measurement: This test was to determine the available latency from the system when it is interrupted or when the task is changed. During the trials, tasks were periodically performed to evaluate the running time consistency and stability. Due to the use of microseconds, which is very important in cases like suspension control systems where minimum missed deadlines results in major performance differences. Vehicle Suspension System: The suspension control system varies the suspension of the vehicle in real-time depending on the driving conditions. Nonetheless, with such systems, feedback and alterations come into play. In order to allow the suspension system to respond as quickly as possible and to keep it operating at the highest level of performance, the RTOS has to allow for the processing of information received from the vehicle's attitude, motion and even the quality of the roadway in question, in real time, commanding all necessary actions. To conclude, when Xenomai is used with the Linux kernel, it does offer a superior real-time performance enhancement compared to non-real time Linux systems. Features such as low-latency context switches and consistent execution time are important for engineers who make car suspension control systems where timing of milliseconds is very important. Therefore, non-RTOS systems are not suitable for applications such as car suspension control which are safety critical because most of the times, they can complete the tasks faster but they cannot guarantee completion of tasks within the specified time. In terms of features, there are many that Xenomai has, but in terms of performance and the tasks it can do, that might be sacrificed. For example, some instead support PREEMPT_RT which is a real time Linux module and allow different levels of performance depending on how much real time operation is needed. This report explores how the deficiencies in the timeliness, predictability and reliability of Linux can be addressed by the use of Xenomai, which makes it suitable for use in embedded systems where real lidar IOs for suspension systems control are needed. The results confirm the assumption that real-time operating systems are able to execute time-critical actions that are necessary for such important systems. [7]. [7]6.5. The research paper titled "The RTOS as the Engine Powering the Internet of Things" gives more details about the relevance of the real-time operating system (RTOS) in the Internet of Things (IoT). In this context, VxWorks, an RTOS, widely recognized for its reliability, ease of maintenance, and security features making it ideal for IoT system deployment was presented. The RTOS in question was VxWorks. The author outlines the concept of VxWorks being an advanced embedded IoT platform due to its real-time operability, quick task scheduling without delays and the need for an efficient structure in order to achieve the full deployment of the IoT devices. It can cater for devices of varying degrees including simple input sensors as well as complex systems capable of running different applications due to its inherent scalability and safety critical designs. As it is illustrated, VxWorks has already been used in over 1.5 billion devices hence giving, the manufacturers, a good environment to build networkingbased systems. Exposition of VxWorks platform in IoT The modularity of VxWorks systems: VxWorks is based on a modular design allowing developers to integrate and enhance middleware, protocols and applications without altering the fundamental kernel. The issue of modularity is critical in IoT where devices should be flexible to adapt to different changes in networks or market. Scalability: VxWorks is scaled for a number of IoT devices from small, lightweight machines to large, complex

systems. This is suitable for many Internets of Things applications because it can adapt to varied memory, power, and computation requirements. Security: VxWorks offers a variety of built-in security options such as tamper resistant code, secure boot and execution mechanisms, and safe data movement amongst others. These properties are vital in ensuring that Android devices, for instance, hundreds of malwares ridden, unauthorized users and other threats are out of reach.[8]

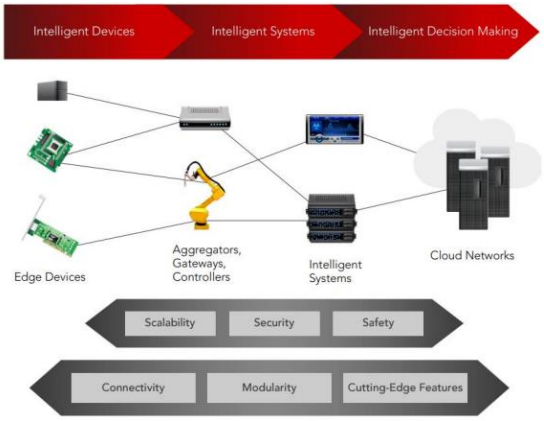

Fig.6. RTOS must adapt to meet the new challenges of IoT[8].

The RTOS supports Networking: Bluetooth, ZigBee, Wi-Fi standards are integrated in IoT operating systems compatible with RTOS, which allows IoT devices to connect and communicate in different network scenarios without hassle. This connectivity is important as there is M2M communication and transfer of real-time data involved in IoT systems. Determinism: VxWorks is recognized for its deterministic credentials thus, one of its main advantages relates to meeting strict scheduling i.e. timing requirements which is very important for IoT systems/applications which are real-time. This ensures that the IoT devices function safely and efficiently wherever deployed especially when those areas are restrictive like the industrial control system or medical device. The avionics of the F-35 lightning II, a fifth generation stealth aircraft manufactured by Lockheed Martin, is powered by VxWorks RTOS whose major focus is reliability and real-time performance during in-flight activities. Thanks to VxWorks which guarantees deterministic operation, there is timely processing of critical inputs from sensors like cameras and radar which is essential for the success and safety of the mission. Its well-defined, integrated, and extensive architecture allows for easy incorporation of newer enhancements and upgrades, while its built-in protections deter threats to the RTOS. In order to ensure seamless interaction with other aircraft as well as ground controls, various communication protocols are supported by VxWorks. This reflects the importance of VxWorks in modern military aircraft as it enhances improved awareness, increased safety and operational capability. Finally, because of such advanced features like minimum latency, multi-core processor support and a well-patterned design, VxWorks RTOS is a good candidate for IoT solutions. Because of its extensibility, safety, and proven reliability, it remains the preferred choice for companies seeking to rapidly deploy effective IoT solutions and yet remain competitive. [8]6.6 The study implements a multithreaded real-time operating system RTOS that was used to control a robotic vehicle. The device was based on the dual core ESP32 microcontroller which is ascribed to the fact that the microcontroller has inbuilt Bluetooth and Wi-Fi connectivity. With this RTOS present in this framework, it can carry out many functions at the same time all spearheaded by controlling the motion of the motors, monitoring sensors, and also implementing communication via Bluetooth with an Android application. This project included real time control by the users of the system employing applications within the environment and also real time control of the system processes employing the Arduino IDE programming platform as well as the MIT app innovator for the mobile application. In terms of task allocation, time allocation and movement control responsiveness within the vehicle, this was all managed using the RTOS. For instance, in the case of a vehicle, the motors of the aforementioned vehicle are controlled using user inputs while at the same time ultrasonic sensors are in the operation of scanning the surroundings for possible obstacles. To avoid these obstacles, the RTOS quickly brings the vehicle to a halt in instances when an obstacle is detected. This system design is more focused on real time operations bearing in mind the RTOS features all completion of all activities performed in time. The other components of the system include a DC-DC converter for providing regulated power, ultrasonic sensors for obstacle detection and an L298N motor driver for motion control of the robotic platform. The RTOS helps to ensure that all the tasks are performed concurrently without interrupting the flow of performance in real time interaction and the whole system design is aimed at performing quite well without any hitches. To finish, a good example of a technique for controlling real-time operations in embedded systems is the use of the RTOS which is a multi-threaded Real Time Operating System framework for ESP32 microcontroller. This innovation ensures opportuneness and steadfastness by empowering concurrent errand execution, especially in

applications that call for human contact and natural observing. This extend illustrates the plausibility of coordination RTOS with versatile apps, in this manner giving a versatile and customizable stage for advance headways in mechanization and mechanical autonomy. [9]

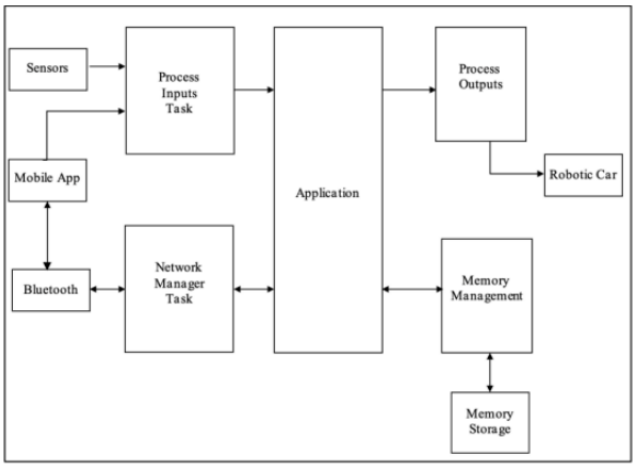

Fig 7. Overall Software Architecture [9]

V. CHALLENGES

There are several very critical challenges one faces in the use of real-time operating systems within modern embedded systems, especially in industrial and event-driven applications. Indeed, one problem is mainly getting around the low resource utilization and high-performance conundrum. RTOSes are designed to run with minimal memory and processing overhead but must ensure timely and predictable execution of tasks. This is particularly challenging in resource-scarce environments, for instance small microcontrollers, as balancing task responsiveness and system efficiency is quite tough. The other important issue lies in a very restricted support for dynamic scheduling in most RTOS implementations. Typical RTOS architectures are highly based on static, priority-based preemptive scheduling that simply is not up to the challenge of handling the inherently nondeterministic nature of event-driven tasks, which causes missing deadlines and less-than-optimal use of resources. This situation is further complicated with added advanced functionalities, such as artificial intelligence or machine learning workloads. These types of jobs are often highly computational in nature and compete with the requirement for high-frequency, time-critical operations like sensor control or even user interface management. Moreover, it is sometimes challenging to scale and achieve flexibility in RTOSes in relation to supporting different demands without significant overheads from the application. For example, extending scheduling policies or adapting energy efficiency mechanisms can add extra memory usage and complexity, which is an unfavorable aspect for light systems. The cross-platform lack of standardization across various RTOS platforms poses compatibility and integration challenges while working on industrial IoT application cases, whereby multiple devices and protocols need to interact seamlessly. Security and reliability are ongoing concerns because RTOSes often operate in safety-critical systems: robust fault tolerance and data protection without degrading performance is demanding. This requires innovations in protocol handling and synchronization as industrial and embedded systems evolve to include more interconnected devices, higher-level communication stacks, and real-time networking. These challenges highlight the need for continuous innovation and optimization in RTOS design to meet the growing demands of real-time and embedded applications.[10]

1. In the case where an RTOS is being used the decision making process about the appropriate software architectural style for the CPS is complex. This is especially true in medical applications for example plasmapheresis equipment. CPS are characterized as those that need timely and accurate responses to some stimuli external to the system since this is much important in circumstances that involve the lives of individuals. The demand for interrupt response time, job switching, and deadlines complicate decision making owing to architectural specificity. The software architecture of the plasmapheresis machine is required to manage a number of components which must be regulated simultaneously and the regulation of each depends on specified time constraints. There two fundamental ways of practicing such systems namely an interrupt-driven model/ or an RTOS-based architecture. Due to serialization, structure by interrupt the architecture may involve problems such as complex nested interrupt control and potential delay in a solving of tasks. Often this architecture requires several distinct timers which is inconvenient and can lead to improper management of critical operations. On the other hand, an RTOS provide a well-defined timing subsystem, and priority scheduling to create a proper way of handling time-sensitive + non-sensitive processes. To optimise the usage of resources and ensure that all the stipulated time frames of important events are met, the RTOS can provide for the apparent parallelism of a number of jobs. Since delay in response could culminate to serious harm to the

patient, this functionality is vital in the execution of medical devices. Thus, the given evaluation proves that the proposed RTOS-based architecture is more dependable and efficient for delivering real-time control in medical applications due to having specific interdependent and accurate timing constraints for the CPS under consideration. [10]

2. The work also examines the challenges that real-time operating systems (RTOS) face while trying to address the requirements of artifcial intelligence (AI) as well as the Internet of Things (IoT) in time-sensitive applications. With increasing numbers of IoT, they generate a lot of data that have to be analyzed on the fly, for instance, in driverless-car and smart-city applications. To ensure dependability and safety the RTOS should ensure that critical operations are done in time. It is always challenging to achieve and even more so when there are more significant hurdles such as network latency and unpredictable data traffic. Moreover, the adoption of the AI algorithms introduces more challenges in terms of the high computational cost, which may be incompatible with the need for real time processes. In order to decrease the latency, the developers are encouraged to optimize these algorithms additionally and use principles of edges computing. Due to these various interconnections these are also real time systems hence there arises the issue of security especially in respect to their data where high levels of security must be placed to ensure that the systems perform optimally while at the same time maintaining the security of the data. Among the various methods of controlling the execution of tasks in RTOS, the Round Robin scheduling algorithm is described in detail in the study. Since it provides a cyclic allocation of time fractions for various activities, it helps significantly in time-sharing of at least several processes. This is very useful in the Internet of Things where several devices need to work on data at the same time. While Round Robin is effective for activities of similar importance in systems with low load and for most real-time applications, there may be situations when it is not suitable, namely, cases if stricter time constraints are required. In summary, the study seeks to establish that in meeting the diverse needs of the current applications, RTOS must super rightly overcome the complexities of data management, algorithms, and security. This is also true as it also recognizes scheduling algorithms such as Round Robin in the achievement of these objectives.

VI. CONCLUSION

Real-Time Operating Systems (RTOS) are important technologies in many areas that require predictable and reliable performance such as robotic cars [9], space missions [5][11], vehicle suspension systems [7] and patient monitoring systems [4]. In such systems, there is always the need for an RTOS to allow multitasking, handle interruptions and respond to events in real time [2]. The systems perform efficiently and reliably and safety was enhanced by the employment of RTOS in these applications. Nevertheless, it comes with a few drawbacks that might hinder its full potential. One of the fears that is primarily important is making sure that the system is immune to soft errors [3]which otherwise would cause the system to crash or device corruption. There are many reasons for soft errors including radiation, power supply fluctuations, and software issues. Therefore, in order to ensure the constant operation of the system, the RTOS has to be designed in such a way that it can identify and correct these errors. Minimizing task switching latency on the other hand is quite a challenge [10], more so where there are strict time constraints on the execution of various operations. To ensure that timeframes are respected, this kind of an RTOS must allow swift and efficient changes from one task to another. It has been demonstrated through research carried out on RTOS case studies based on open sources such as Linux and OSEK, that these challenges can be tackled with resolve and good performance can still be achieved in spite of these challenges. For example, Linux is used in a variety of platforms that include embedded systems and Android mobile devices while OSEK is used in automotive applications such as engine control system and anti-lock braking systems. The ability to observe and develop Real-Time Operating Systems (RTOS) that meet the characteristics of predictability, reliability, and efficiency has been illustrated. In addition, the use of RTOS in advancing technology such as sensor networks, and the Internet of Things (IoT) has highlighted the extent to which they are effective in real time processing and decision making. Thus while RTOS will provide the necessary timing and synchronization mechanisms in the sensor networks to ensure that data is transferred reliably and efficiently, it will also offer the necessary framework in IoT [12]to handle and analyse large amounts of data generated by various sensors and devices. The incorporation of RTOS in these applications is marveled due to its capability of creating new innovative applications and services which in turn can re-structure business processes and social activities. Several approaches such as the use of fault tolerant systems, migration hints [15], Bayesian

networks[5], and advanced scheduling algorithms have been proposed in literature to enhance the efficiency, robustness and reliability of RTOS. Migration hints may assist cut down on the task migration costs; such costs are otherwise disadvantageous to the system by causing delays, while Bayesian networks help in modelling evaluating or even forecasting failures, errors, and aberration in the complex systems. Imagine being predicted by advanced scheduling techniques that will make it possible for us to add, detect imperfections in a system and also carry out corrective actions if things went wrong within the defined time frame [14]. These approaches make the implementation and performance of RTOS enhancement providing the necessary requirements of the highest of all real-time applications. The studies presented in this paper demonstrate that research and development endeavours aimed at overcoming barriers concerning the adoption of real-time operating systems RTOS are of utmost importance. Additional research and technological interventions are required to improve the efficiency, reliability, and overall performance of RTOS, as these systems are still critical in the evolution of real-time systems.

REFERENCES

[1] Barbareschi, M., Barone, S., Casola, V., Montone, P., & Moriconi, A. (2022). A Memory Protection Strategy for Resource Constrained Devices in Safety Critical Applications. A Memory Protection Strategy for Resource Constrained Devices in Safety Critical Applications. https://doi.org/10.1109/icsrs56243.2022.10067350

[2] Ivanov, B. I., Hotmar, A., Ivanov, I. E., & Georgieva, D. (2023). A Software Architecture Selection for Cyber-Physical System. A Software Architecture Selection for Cyber-physical systems. https://doi.org/10.1109/comsci59259.2023.10315884

[3] Lencioni, L. R., Loubach, D. S., & Saotome, O. (2022). An Application-Specific Real-Time Operating System Towards Safety-Critical Requirements: a Case Study. 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), 1–10. https://doi.org/10.1109/dasc55683.2022.9925758

[4] Nayak, S., Sharma, Y. K., & Student, R. N. (2023). An Optimized Round Robin Central Processing Unit Scheduling Algorithm with Dynamic Time Quantum. An Optimized Round Robin Central Processing Unit Scheduling Algorithm With Dynamic Time Quantum, 3, 1482–1490. https://doi.org/10.1109/smarttechcon57526.2023.10391489

[5] Moghadam, V. E., Meloni, M., & Prinetto, P. (2021). Control-Flow Integrity for Real-Time Operating Systems: Open Issues and Challenges. Control-Flow Integrity for Real-Time Operating Systems: Open Issues and Challenges. https://doi.org/10.1109/ewdts52692.2021.9581003

[6] Taji, H., Miranda, J., Peón-Quirós, M., Balasi, S., & Atienza, D. (2023). Dynamic Scheduling for Event-Driven Embedded Industrial Applications. Dynamic Scheduling for Event-Driven Embedded Industrial Applications, 1–6. https://doi.org/10.1109/vlsi-soc57769.2023.10321845

[7] Sudhakar, A., C, R., & S, A. (2022). Implementation architecture of IEC 60870-5-103 communication protocol on arm platform running on RTOS for industrial IEDs. 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA), 3, 80–86. https://doi.org/10.1109/icirca54612.2022.9985497

[8] Veeramanikandasamy, T., Babu, P. R., Devendiran, S., & Aravind, N. (2023). Internet of Farming Things and RTOS based Robotic System for Water Quality Monitoring and Fish Feeding in Freshwater Aquaculture. Internet of Farming Things and RTOS Based Robotic System for Water Quality Monitoring and Fish Feeding in Freshwater Aquaculture. https://doi.org/10.1109/icccnt56998.2023.10308205

[9] Forlin, B., Chen, K., Alachiotis, N., Cassano, L., & Ottavi, M. (2024). Lightweight Instrumentation for Accurate Performance Monitoring in RTOSes. Lightweight Instrumentation for Accurate Performance Monitoring in RTOSes, 1–2. https://doi.org/10.23919/date58400.2024.10546790

[10] Pradhani, J., D, N., Nidhi, N., Saran, C. S., Goni, A., Mallidu, J., Itagi, A. R., & Patil, A. (2024). Prototype Development for Water Leakage Monitoring System with RTOS Implementation. Prototype Development for Water Leakage Monitoring System With RTOS Implementation, 1–7. https://doi.org/10.1109/icdcot61034.2024.10515556

[11] Badawy, A. A., Hassan, M. A., Hassaballa, A. H., & Elhalwagy, Y. Z. (2024). Real Time Integration GPS with Gyro-Compassing Using Two Cascaded EKF with Free RTOS. Real Time Integration GPS With Gyro-compassing Using Two Cascaded EKF With Free RTOS, 7, 307–314. https://doi.org/10.1109/icci61671.2024.10485034

[12] Delgado, R., Jo, Y. H., & Choi, B. W. (2022). RT-AIDE: a RTOS-Agnostic and Interoperable development environment for Real-Time Systems. IEEE Transactions on Industrial Informatics, 19(3), 2772–2781. https://doi.org/10.1109/tii.2022.3182790

[13] Serino, A., & Cheng, L. (2020). Real-Time Operating Systems for Cyber-Physical Systems: Current Status and Future Research. Real-Time Operating Systems for Cyber-Physical Systems: Current Status and Future Research.

https://doi.org/10.1109/ithings-greencom-cpscom-smartdata-cybermatics50389.2020.00080
[15] S, S. M., & S, N. G. (2021). A survey on different real time operating systems. International Journal of Engineering and Advanced Technology, 10(5), 221–223. https://doi.org/10.35940/ijeat.e2762.0610521
[16] Bini, E., Chantem, T., Childers, B., & Mosse, D. (2022). IEEE TC Special issue on Real-Time Systems. IEEE Transactions on Computers, 72(1), 1–2. https://doi.org/10.1109/tc.2022.3227228
[17] Rico, R., Rico-Azagra, J., & Gil-Martinez, M. (2022). Hardware and RTOS design of a flight controller for professional applications. IEEE Access, 10, 134870–134883. https://doi.org/10.1109/access.2022.3232749