



XIRAC: an optimized product-oriented near real-time operating system with unlimited tasks and an innovative programming paradigm based on the maximum entropy method

Alireza ZIRAK

Photonics and Quantum Technologies Research School, Nuclear Science and Technology Research Institute, Tehran, Iran

E-mail: zirak@um.ac.ir

Received Feb. 12, 2024; Revision accepted Aug. 5, 2024; Crosschecked

Abstract: In the fiercely competitive landscape of product-oriented operating systems, including the IOT, efficiently managing a substantial stream of real-time tasks, coexisting with resource-intensive user applications embedded in constrained hardware presents a significant challenge. Bridging the gap between embedded and general-purpose operating systems, we introduce XIRAC, an optimized operating system shaped by information theory principles. XIRAC leverages Shannon's Information Theorem to regulate processor workloads, minimize context switches, and preempt processes by maximizing system entropy tolerance. Unlike prior approaches that apply information theory to task priority alignment, the proposed method integrates maximum entropy into the core of the real-time operating system and scheduling algorithms. Subsequently, we optimized numerous system parameters by shifting and integrating commonly used unlimited tasks from the application layer to the kernel. This paper describes the many advantages ushered in by this paradigm shift, offering improved system performance, scalability, and adaptability. A new application-programming paradigm, termed "object emulated programming" has emerged from this integration. Practical implementations of XIRAC in diverse products have revealed additional benefits, including an enhanced learning curve, elimination of library functions and threading dependencies, optimized chip capabilities, and increased competitiveness in product development. We provide a comprehensive explanation of these benefits and explore their impact through various real-world use cases and practical applications.

Key words: Task scheduling; Information theory; Embedded Real-Time Operating Systems; Maximum entropy; Load balancing

<https://doi.org/10.1631/FITEE.2400102>

CLC number:

1 Introduction

Real-Time Operating Systems (RTOS) have long grappled with stringent constraints, aiming to enable the execution of unpredictable tasks to

meet their deadlines, while ensuring seamless interaction with underlying hardware resources (Wang, 2020; Zirak, 2023). These limitations are deeply rooted in the conventional structures of operating systems and schedulers, which necessitate software development through gradual, unpredictable evolution, relying primarily on constrained tools such as functions and threads.

ORCID: Alireza ZIRAK, <https://orcid.org/0000-0002-0346-7998>

© Zhejiang University Press 2025

The existence of these challenges, combined with hardware constraints, has contributed to a noticeable schism within the domain of computer systems, resulting in the classification of two distinct classes: general-purpose computers and embedded systems. General-purpose computers are characterized by operating systems that provide a versatile environment, enabling users and machines to collaborate in executing a diverse array of tasks. Conversely, an embedded operating system is engineered for highly specific functions, often requiring minimal or no user intervention (Wang, 2020).

The intrinsic simplicity of embedded systems, characterized by their modest functionality, hardware limitations, low power consumption, and compact size, has fueled their widespread adoption. The rapid progress of embedded systems, especially the IOT (internet of things), has blurred the lines between sectors, where in many cases the terms general-purpose computer and embedded system have become mishmashed to mean the same thing. Thus, embedded systems have flourished, driven by diverse user requirements and intense product-oriented competition. This competition prompts a critical question: What lies ahead in the realm of technology?

The answer to this question hinges on the trajectory of computer hardware development, as highlighted by Dan Reed, a keynote speaker at an ISC conference. He noted that while Moore's Law is approaching its limits with the advent of 4-nm technology, new computing opportunities are emerging, particularly with the rapid progress of quantum computing. He emphasized the importance of architectures to continue pushing boundaries, even as classical methods reach their peak. Though these estimates may involve some degree of approximation, the core message has been affirmed by scientific consensus. Quantum computers, however, do not aim to replace their classical counterparts. Classical computers have

unique attributes that quantum computers may find challenging to replicate - the ability to store persistent data, fault tolerance, gate fidelity, availability of diverse resources, and cost-effectiveness, all of which render classical computers superior for a vast array of prevalent tasks.

Consequently, even as the need for augmenting classical computer capabilities remains acute, particularly in the realm of embedded systems, the outlook for scaling and speeding up their hardware resources is not promising. Furthermore, from the perspective of quantum computing, it is imperative to recognize that a single additional qubit has the potential to double computational power. As the number of qubits increases, quantum computers demonstrate exponential superiority over their classical counterparts, heralding the dawn of quantum supremacy. This beckons companies to vie for the production of robust quantum supercomputers with significantly greater stable qubits and gates. However, this quest also creates a surge in technological challenges and maintenance costs (Fang et al., 2020; Min-Allah et al., 2012; Zirak, 2023). Consequently, even if quantum computers surmount these challenges, sharing access to a costly quantum supercomputer will only be feasible through cloud technology.

In light of these constraints, optimizing scheduling in future robust operating systems, encompassing classical embedded systems and visionary quantum supercomputers, assumes paramount importance. In our previous work (Zirak, 2023), we unveiled XIRAC-Q, a quantum operating system designed for ion trap-based quantum supercomputers, which was emulated with FPGA. This innovative system leverages principles derived from information theory to effectively schedule tasks and optimize performance. In the realm of classical computing, the proposed structure has exhibited exceptional robustness, enabling the rapid development of a wide array of products and projects from concept to completion

over the past 20 years. Built on maximum entropy principles, this classical system was instrumental in meeting diverse technological demands long before we introduced its quantum counterpart. Its practical advantages, adaptability, and efficiency in real-world applications have made it far more comprehensive and valuable compared to the quantum version. In this study, the primary objective was to maximize the number of active or ready near real-time tasks while optimizing CPU usage, particularly by increasing the number of orthogonal tasks to address the most critical and pervasive needs. To achieve this, we propose a novel paradigm that goes beyond traditional general-purpose and embedded systems, forming the core of this article's focus. Additionally, the entire operating system can be executed as a single thread within another operating system, such as running it as one of the tasks on a Raspberry Pi. By leveraging maximum entropy principles, we have successfully migrated unlimited application-level tasks to the kernel, making it highly effective in small to medium-scale processors, such as those used in smart home systems and consumer applications. The simplicity and interactivity of the application layer have also proven invaluable for educational purposes, providing a rare hands-on learning experience.

Despite the extensive body of literature on scheduling algorithms and task parameter adjustments, an operating system's tolerance and adaptability to task disarray pose persistent challenges, particularly from a CPU perspective. Achieving effective optimization also necessitates more than optimal-tuning scheduling algorithms and task parameters; it demands a holistic approach aimed at maximizing system utilization and performance, especially in environments with a growing multitude of processes and tasks. From a scheduling perspective, this involves not only adjusting algorithm parameters but also establishing a closer and more dynamic coordination between the scheduler and hardware. This coordination ex-

tends beyond conventional roles of library functions, requiring a strategic reassessment of their influence. It also entails a deliberate reconsideration of existing constraints, such as occasionally allowing tasks to miss deadlines to avoid unnecessary transitions to diverse circuits and viewing all task priorities as non-critical. Importantly, this novel perspective holds promise for advancements in diverse domains, potentially leading to innovative programming paradigms and suggesting transformative alterations to CPU architectures. This holistic approach seeks to address the root challenges in operating system performance, pushing the boundaries of traditional paradigms and opening avenues for groundbreaking developments.

This study takes advantage of the capabilities offered by a resilient operating system to effectively manage an extensive range of computational tasks. In contrast to traditional operating systems, where the burden is largely placed on compilers and programmers during compile time, our approach introduces a groundbreaking method to enhance scheduling of an infinite number of tasks and messages within a robust near real time operating system. By using the entropy maximization technique and harnessing the tenets of information theory, our innovative solutions have been meticulously tailored to cater to the needs of robust operating systems.

Experimental results reveal a multitude of advantages over prevailing scheduling solutions, particularly when dealing with a growing number of tasks and processes. These findings underscore the exceptional and remarkable performance of the proposed method, showcasing its potential to tackle product-oriented development challenges, especially when faced with an ever-increasing number of processes and tasks.

This article's structure is outlined as follows: Section 2 delves into the requirements of operating systems, considering hardware factors as well.

Section 3 reviews foundational research on popular scheduling algorithms, laying the groundwork for our innovative methodologies and demonstrating the key advancements that distinguish our proposed approach. Section 3 reviews foundational research on popular scheduling algorithms, laying the groundwork for our innovative methodologies and demonstrating the key advancements that distinguish our proposed approach. This section also presents the necessary background information and derivations for our solutions. In Section 4, we introduce fundamental concepts, particularly within the domain of information theory, and delve into the XIRAC architecture. This includes discussions on near real-time scheduling methods of numerous tasks, the messaging system and communication strategies, all based on maximum entropy, leading to a novel programming paradigm. Section 5 assesses the performance of our proposed operating system, while Section 6 provides a summary of the article.

2 Motivation

In light of the challenges at hand and the viewpoint of this article, a fundamental question arises: "How can we enhance the scheduling of numerous near-real-time tasks within a resource-constrained system, while implementing a robust operating system?" This question unites two critical processing technologies-embedded product-oriented operating systems and cloud-based quantum supercomputer operating systems-under the same umbrella.

2.1 Robust task scheduling and constraints

In the context of minimal systems, various commercial platforms have emerged to streamline microelectronic system design, particularly when hardware resources are scarce. Platforms like ARDUINO, BEAGLEBONE, and RASPBERRY-

PI offer compelling features and have garnered substantial user bases (Kondaveeti et al., 2021). Nevertheless, these platforms often provide users with a collection of disparate, loosely correlated library functions tailored to specific hardware components. As a result, achieving the key attribute of cost-effective and competitive products, involving the seamless integration of extensive capabilities in real-time fashion, remains elusive. Typically, these types of solutions find application mainly in laboratory and educational settings. However, in product-oriented and practical scenarios, expanding the size of user applications, particularly in terms of the number of non-critical tasks, becomes imperative and inevitable. Yet, this expansion introduces the challenge of resource contention among tasks, potentially leading to data vulnerabilities, application crashes, or substantial periodic jitter. Therefore, effective real-time task scheduling becomes indispensable to attain optimal performance.

In the realm of real-time system design, we grapple with two pivotal issues: (i) establishing criteria for adjusting task parameters, and (ii) selecting a scheduling algorithm to maximize system utilization (Wang, 2017; zirak, 2023). From a scheduling viewpoint, modern real-time systems prioritize aspects like meeting task deadlines, with a focus on prioritizing critical task deadlines over fewer essential ones. This preference leads to the adoption of fixed priority scheduling mechanisms such as Rate Monotonic (RM) due to their simplicity and adaptability, particularly during periods of heavy system load or when managing static tasks. In contrast, dynamic scheduling algorithms like Earliest Deadline First (EDF) come into play under lighter system loads, although they may introduce computational overhead owing to the need for frequent ready queue sorting, and they may exhibit instability (Wang, 2020).

In each of these instances, task periods and deadlines are typically established during the de-

sign phase and remain constant throughout the system's operation. However, task execution times exhibit considerable variation. The upper bound of task execution time i.e., Worst Case Execution Time (WCET), plays a pivotal role in scheduling algorithm calculations (Fang et al., 2020; Min-Allah et al., 2012). The main difficulty is that many scheduling-related computations, such as the utilization factor, rely on an over/underestimated parameter due to the influence of unpredictable environmental conditions on task execution times. Therefore, relying on parameters that have the nature of a random variable, rather than deterministic values, introduces substantial estimation error, particularly as the task accumulation increases.

Within multitasking RTOS, especially in robust applications characterized by a significant upsurge in task numbers and the size of user applications, there is a looming possibility of the operating system missing tasks. This situation can result in application crashes or, at minimum, perceptible periodic or delayed jitter. Contrary to current practices, relying solely on library functions and dividing the kernel and applications into separate, independent blocks may not be an optimal solution. The limitations imposed by hardware and resource constraints further exacerbate these issues, making them particularly prominent in numerous request scenarios.

2.2 Exploring the foundations of novel approaches

When we direct our attention to the concept of "load balancing", it becomes evident that a considerable portion of research papers in this domain mainly use techniques that rely heavily on measurements of CPU utilization, which are closely associated with the WCET (Worst-Case Execution Time) (Avan et al., 2023; Fang et al., 2020). However, depending solely on WCET poses notable challenges. These include the pro-

pensity for overestimation or underestimation due to a strong dependence on random environmental conditions and limitless requests with a lack of quantitative parameters. This issue is particularly pronounced in product-oriented designs owing to the inherent volatility of requirements. Therefore, the primary goal is to maximize the number of active or ready near real-time tasks while optimizing CPU usage. On the other hand, minimizing the reliance on WCET is crucial for efficient CPU utilization in product-oriented competitive projects. To achieve this, we propose migrating an unlimited number of non-critical, common application-level tasks to the kernel.

These constraints necessitate the introduction of a new paradigm applicable to all computer systems, introducing a third category we named it the "Robust Architectural Core". This nomenclature reflects the imperative need for a robust kernel that efficiently correlates the unpredictable, high demands of clients with the constrained hardware and bandwidth resources, all converging into a unified architectural core during runtime. In contrast to conventional methodologies, which involve configuring different components of the kernel and application software through independent and randomly accessed library functions or APIs, this cohesive approach stands out. This vacancy forms the separate boundary that distinguishes the Robust Architectural Core from the domain of general-purpose computers and embedded systems.

Furthermore, achieving the optimal integration and scheduling of a broad spectrum of diverse tasks within the kernel is of paramount importance. To optimize overall system performance, we adopt a mathematical objective function to model the computing system. The key to this optimization is identifying the degrees of freedom that define the RTOS's capabilities, with task request parameters and their scheduling becoming primary variables in optimization. The

core criterion for this optimization must be the system's overall performance.

In this investigation, we embraced an informative representation of a diverse array of tasks and requests, strategically used to optimize and assess the efficacy of a streamlined scheduling method. Through the systematic maximization and regularization of information content, with a specific emphasis on elevating entropy toleration, we achieved an elevated threshold for managing the disordering of tasks and messages, thereby enhancing overall project manageability. This intricate process involves dynamically deriving a probability distribution tailored to our present understanding of the dynamic operations within subsets of the system. Incorporating measurable data such as the variance and the mean of events or requests and then drawing upon the fundamental principle of maximum entropy, we constructed a distribution that minimizes assumptions while showcasing the most unforeseen behavior in relation to our scheduling parameters.

The impressive outcomes achieved by our method can be attributed to the shared characteristics inherent in various processes, thus enhancing its informative aspect. Over several years of technical and research activity, we have developed a refinement method that culminated in the creation of an open-source operating system kernel. This endeavor has been instrumental in the realization of numerous diverse products.

This approach, using optimal load balancing by entropy maximization of the overall system, fosters the development of robust applications with simplified interfaces. The system stands in contrast to prevalent operating systems where the task scheduling responsibility indirectly rests with programmers and compilers, often leading to an uncoordinated and gradual growth of concurrent operations via threading or other prevalent real-time programming techniques. Achieving this approach involves adjusting scheduling param-

eters, improving scheduler-request correlation, using stochastic parameters and a-priori information, and eliminating unnecessary constraints.

To achieve these goals, over 15 years we have developed and refined an embedded system platform and operating system, leveraging Shannon information theory principles, a-priori information and the Nyquist-Shannon sampling theorem. We have developed an operating system structure called XIRAC (Extreme Informative Robust Architectural Core), which effectively integrates common real-time features into the kernel while minimizing CPU utilization. Note that we previously emulated a somewhat similar structure in a quantum computer (XIRAC-Q), leading to the current implementation in the classical counterpart (Zirak, 2023). This perspective enhances the learning curve for designers, developers and educational initiatives, while offering greater flexibility and performance for the final product. The framework introduces a novel paradigm in programming, which we have termed "object emulated programming". This method simulates object-oriented structures at a deeper system level, fostering innovation and efficiency in software development.

In this context, we represent every process as a random set of DAGs (Directed Acyclic Graph), where each DAG is scheduled according to requests with random time scaling. For every DAG, the nodes symbolize strictly periodic execution paths, e.g., sets of tasks or messages, and the directed edges indicate dependency (Jyotsna et al., 2024; Zafari et al., 2019). Each process, communication protocol, or application can be conceptualized as a periodic execution path graph. Given that each DAG event follows a Poisson distribution, with an infinite number of tasks, the aggregate behavior of a large set of DAGs approximates a Gaussian distribution with defined probabilistic parameters.

To highlight the effectiveness of our schedul-

ing methodology, this paper incorporates findings derived from extensive data collected over several years across numerous products crafted through this operating system. These results are also compared with similar products to provide a comprehensive performance analysis in low CPU utilization scenarios. Additionally, we will present performance outcomes represented on logarithmic scales, offering a more intuitive grasp of the proposed operating system. Through these results, we aim to highlight the common features that synergistically contribute to the improved performance observed in our embedded operating system.

3 Background information

3.1 Relevant prior research

In addressing the challenges posed by common scheduling constraints, such as WCET and deadline, many practical applications based on minimal systems encounter limitations when using real-time operating systems or even minimum abstraction layers. Consequently, designers often resort to bare-metal programming, a practice exemplified by popular platforms like ARDUINO. However, these platforms face a challenge rooted in their limited expandability, restricting their utility to mainly amateur and laboratory applications.

Previous studies have attempted to use criteria other than WCET for task scheduling, with some exploring variables rooted in information theory, notably entropy (Abohamama et al., 2022; Avan et al., 2023; He et al., 2015; Li et al. 2018; Rincón et al., 2018; Rincón et al., 2017; Sharma et al., 2013). However, the advances in scheduling algorithms resulting from such efforts have been marginal. For instance, in (Avan et al., 2023; Rincón et al., 2018; Rincón et al., 2017), innovative solutions using entropy for dynamic task priority scheduling are presented, prioritizing tasks with the highest information content per studied

interval. Yet, in all these endeavors, the entropy component exhibits a sole dependency and remains proportional to the utility factor.

In (Rincón et al. 2017), they introduced the RTS-RT method for comparing task information, yet the information components of priority criteria, i.e. H_{sys} and d_t , are uniform for all tasks. Consequently, the entropy in these methods lacks a discernible effect on priority comparison, and the criteria employed for parameter selection align closely with those of prevalent methods for calculating utility factors or probabilistic occurrences.

In Li et al. (2018), they proposed a hardware-oriented task partitioning and scheduling algorithm that considers race conditions, using an entropy maximization method. While their approach considers uncertainty as a probabilistic occurrence, leading to an uncertainty matrix analysis, its high computational complexity and strong external condition dependency hinder its broader applicability. Similarly, in He et al. (2015) an uncertainty model was introduced for race conditions but did not develop task scheduling algorithms. These proposed solutions are claimed to enhance system (CPU) performance. However, from an information theory perspective, they mainly reduce embedded information within the time division (source entropy) without addressing the system's capacity (channel entropy). In the realm of optimization, there has been a notable emphasis on scheduling for safety-critical systems in recent years (Jyotsna et al., 2024). However, it is crucial to highlight that this attention predominantly centers around critical systems. Conversely, the core focus of this article lies in non-critical systems, which forms the basis of our exploration.

In our previous study, Zirak (2023), we employed a maximum entropy approach to theoretically optimize quantum tasks, focusing on ion-trap-based quantum computers. However, this work was primarily theoretical and did not involve a classical operational system with tangible,

real-world output data. Tackling the optimization challenge in this domain holds the potential to unveil compelling and intriguing results, particularly in product-oriented systems. Examples include robust operating systems adept at managing infinite tasks, all within the framework of the proposed object emulated programming paradigm. Our practical results, as detailed in this article, highlight how products and methodologies developed over two decades have benefited from a maximum entropy-based distribution framework. This approach has proven critical for optimized resource allocation and enhanced system flexibility, reflecting an even distribution of potential states. Such adaptability supports robust performance in product-oriented embedded systems, especially within competitive applications that involve complex, real-time tasks.

3.2 Fundamental concepts

Shannon (Shannon, 1948) introduced the concept of information entropy to quantify the uncertainty, or “surprise,” associated with the occurrence of a random variable. For a discrete random variable T , representing all active tasks set with potential values $\{t_1, \dots, t_n\}$ and probability mass function p_i , entropy $H(T)$ quantifies the distribution of tasks and the likelihood of each state, encapsulating both as a single metric. This entropy H can be defined as:

$$H(T) = E[I(T)] = E[-\log(P(T))] = -\sum_{i=1}^n p_i \ln(p_i). \quad (1)$$

where E the expectation operator, and $I(T)$ signifies the information content of T , which is conceptually akin to the system’s tolerance (CPU capacity) for handling task events or requests. Here, $p_i = P(T = t_i)$ represents the probability of each task state, with $H_n(p_1, p_2, \dots, p_n) = H(T)$ expressing the entropy in terms of the distribution probabilities.

Maximum entropy posits that when statistical constraints such as mean, standard deviation or other momentums of random variables are available, the probability distribution representing the current state of knowledge with the highest information entropy is considered (Fampa et al., 2024; Khosravi et al., 2023; Lisitsin et al., 2024). Maximizing entropy implies the highest level of surprise and the fewest satisfied assumptions.

To solve for maximum entropy distributions, the problem is typically formulated as maximizing the entropy function with multiple constraints such as mean and standard deviation. Constrained optimization problems like this are commonly addressed by leveraging Lagrange multipliers (Khosravi et al., 2023). To integrate essential constraints within a single scheduling function, we employ a Lagrange multiplier approach, incorporating factors for workload and task priority. An expanded Lagrange multiplier function \mathcal{L} to maximize entropy H under specified constraints is expressed as:

$$\mathcal{L}(p_1, \dots, p_n, \lambda_1, \lambda_2) = -\sum_{i=1}^n p_i \ln(p_i) + \lambda_1 (\sum_{i=1}^n p_i t_i - \mu) + \lambda_2 (\sum_{i=1}^n p_i r_i - R) + \lambda_3 (\sum_{i=1}^n p_i - 1) \quad (2)$$

where, λ_1 and λ_2 are the Lagrange multipliers tied to the statistical constraints μ (e.g., average of tasks requests) and resource limit R (e.g., processing capacity and memory), respectively. Here, t_i and r_i denote the computational load and resource requirements per task. By setting all partial derivatives of \mathcal{L} to zero, we derive a system of equations to p_i values that maximize entropy under the constraints. These constraints could reflect the computational limits of a microcontroller (e.g., CPU cycles, available memory) or statistical properties such as the mean and variance of task requests, allowing for optimization within the device’s physical capabilities and workload characteristics.

As a straightforward example, in a general product-oriented operating system within a standard proces-

sor environment, where only the average task request rate μ is known—while variance and specific resource requirements remain undetermined—the corresponding distribution that maximizes entropy under these limited constraints is given by:

$$Pr(T = t_i) = S \times r^i \quad \text{for } i = 1, \dots, m \quad (3)$$

In this equation, the constants r and S are positive and the expected value of a random variable must be μ . The constants determined by the conditions that the probabilities sum to one. In this case, if each task request or job is treated as a random variable indexed by its repetition rate (e.g., $t_i = i$, where i is the index of each task), then:

$$S = \frac{1}{\mu-1}, r = \frac{\mu-1}{\mu}. \quad (4)$$

Several key characterizations and expressions (Schäffler, 2024; Scharfenaker and Yang, 2020) are formulated, drawing upon foundational functions and derivations rooted in entropy principles. See “foundational functions and derivations” in the supplementary materials for details. The proposed methods and underlying theories are also derived from these principles, providing a basis for critical derivations and implications. Based on the discussions in this paper and the supplementary material, we first outline the inferred theories and their key corollaries;

Theory 1: the future will be led by product oriented embedded or quantum operating systems with an unprecedented number of tasks.

Theory 2: the succeeding generation of embedded systems and quantum supercomputers necessitates the effective management of an extensive array of diverse requests in near real-time, facilitated by a robust and optimized operating system.

Justification 1, 2: increased diversity in user needs and heightened competition among manufacturers have raised public expectations for em-

bedded systems.

Corollary 1: in a resilient and robust operating system, the quantity of tasks should approach infinity in a near real-time fashion, underscoring the need for scheduling optimization.

Corollary 2: To achieve optimal scheduling in a high-efficiency operating system, it is essential to migrate the majority of tasks, particularly frequently requested and common ones, from the application layer to the operating system’s kernel. By integrating and scheduling these tasks directly within the kernel, overall entropy can be maximized, effectively distributing system states and reducing reliance on numerous nested, unoptimized user-level threads. Let T_{app} be a random variable representing tasks in the application layer, and T_{ker} represent the corresponding tasks in the kernel. By defining task transfer as $T_{app} \rightarrow T_{ker}$, the kernel scheduling function that maximizes entropy over the task space can be represented as:

$$\max_{kernel} \{H(T_{app} \rightarrow T_{ker})\} \quad (5)$$

4 XIRAC architecture

4.1 Fundamental structure of the scheduler

From a CPU point of view, whether instructions are simple NOPs (No Operations) or intricate and efficient algorithms, the impact on factors like power consumption or performance remains indifferent. Examining both scenarios through the perspective of information theory, reveals a shift in system entropy in such a way that the information state changes from a minimum to a maximum value. Simply put, maximizing system tolerance from the perspective of entropy indicates a greater capacity to accommodate disordered tasks and messages, ultimately improving project manageability.

The entropy of a system equipped with a Real-Time Operating System (RTOS) can be quanti-

fied by analyzing the entropy of task sets distributed across the kernel and application layers. Let t_i represent i^{th} periodic task and $t_{i,j}$ denote the j^{th} instance (job) of t_i . Define the random variable T to encompass all jobs of all tasks. Each task t_i is characterized by a probability p_i , execution time τ_i , and resource demand r_i . The goal is to maximize the system's entropy $H(T)$, to achieve an optimal task distribution in the kernel, minimizing timing overhead while enhancing scheduling efficiency in the following distinct scenarios:

(i) Unconstrained job Scheduling: For a robust and adaptive scheduling system, assume all jobs are independent and equally likely, reflecting a scenario with no predefined priorities or parameters. Shannon's information theory implies that maximal entropy is achieved under equiprobable distribution:

$$\arg \max_{p(T=t_{i,j})} H(T) = \frac{1}{N}, \quad \forall i, j \quad (6)$$

where N is the total number of jobs in a hyper-period.

(ii) Semi-Constrained Tasks Scheduling: Unlike individual job scheduling, tasks often possess unique statistical parameters based on their nature and requests. For instance, tasks may vary in frequency, execution time, or resource demands. By incorporating such parameters, a more realistic scheduling model can be developed. Define a task set $T = \{t_1, \dots, t_m\}$ arranged by frequency, and let $C = \{c_1, \dots, c_K\}$ denote task requests or calls, mapped to expected periods $P = \{p_1, \dots, p_K\}$ by:

$$f: C \rightarrow P. \quad (7)$$

The expectation $E(P) = \mu$ represents the mean task period. For example, if $\mu = 2$ the corresponding maximum entropy distribution is:

$$\Pr(T = t_i) = u(t_i) = \left(\frac{1}{2}\right)^i, \quad i = 1, \dots, m. \quad (8)$$

The utilization factor of task i , represented by $u(t_i)$, signifies the likelihood that it is currently running. This binomial distribution reflects tasks occurrence probabilities proportional to their repetition rates.

(iii): Integrating tasks from application layer to the kernel: When additional statistical constraints, such as variance or higher-order moments, are considered, the task distribution adapts to these parameters. Depending on the application, the resulting distributions might include Poisson, normal, or other forms. For instance, a Poisson distribution is appropriate for scenarios where tasks are not equally probable, resulting in tasks with unequal durations. However, when the number of tasks tends to infinity, the Poisson distribution approaches a normal distribution, characterized by its mean and variance.

Migrating unlimited tasks from the application layer to the kernel enhances system entropy by facilitating more efficient scheduling at the kernel level. The stochastic parameters influencing this integration include resource demands, task requests, and execution times. In complex and dynamically changing scenarios, traditional analytical methods may be insufficient. Instead, numerical optimization techniques, such as Lagrange multipliers, as shown in Eq. (2), can be employed to determine the optimal probability distribution of kernel tasks. In this equation, differentiating \mathcal{L} with respect to p_i and setting the derivatives to zero, maximizes the entropy $H(T_{ker})$ under the given constraints. The derived probability distribution for optimal task scheduling is (See "Solving probability distribution for optimal task scheduling" in the supplementary materials for proof):

$$p_i = \frac{\exp(\lambda_1 \tau_i + \lambda_2 r_i)}{\sum_{j=1}^n \exp(\lambda_1 \tau_j + \lambda_2 r_j)}, \quad (9)$$

where λ_1 and λ_2 are the Lagrange multipliers associated with timing and resource constraints,

respectively. These multipliers can be computed by solving the system of equations derived from the constraints.

By integrating tasks directly into the kernel, these parameters can be statically defined by the operating system programmer. While this approach is more complex than typical application programming, it significantly improves the efficiency and performance of the operating system. Kernel programming focuses on defining nested tasks and sub-tasks with irregular but highly optimized structures for effective scheduling, whereas application programming prioritizes user simplicity, often sacrificing system-wide efficiency.

Introducing threads in user applications primarily simplifies application programming for non-expert users. However, it's crucial to distinguish between application-level and OS-level programming. While operating system programming should prioritize optimal scheduling, even at the expense of complexity, this increased complexity can paradoxically lead to greater simplicity at the application level. By offloading complex scheduling decisions to the kernel, we can enable the development of new programming paradigms, such as object-emulated programming. This paradigm offers a more intuitive and streamlined approach to application development, potentially revolutionizing the way we write software.

Previous work (Zirak, 2023) optimized the runtime scheduling of quantum tasks for user applications on a classical computer. This study shifts the focus to optimizing the scheduling of OS tasks at compile time, ensuring static efficiency for system-critical operations. By implementing kernel-level scheduling optimized using Eq. (9), operating systems can achieve significant performance gains in real-time and resource-constrained environments.

(iv) Minimum Informative Jobs Execution Time (MIJET): in this scenario, the goal is to min-

imize the WCET (Worst Case Execution Time) of all task instances (jobs) while ensuring the system maintains a "minimum informative" state. This is achieved by increasing the number of jobs to a threshold that maximizes the system's entropy. We refer to this parameter as the *Minimum Informative Jobs Execution Time (MIJET)*, is mathematically defined as:

$$\arg \max_N H_N(T) = \Lambda, \text{ where } H_\Lambda(T) \geq H_{\Lambda+1}(T). \quad (10)$$

Here, $H_N(T)$ denotes the entropy of N jobs within one hyper-period, while Λ represents the maximum number of jobs satisfying the MIJET condition. Notably, the right side of the equation signifies that the entropy of the system will not increase further by adding more tasks beyond Λ . This result stems from the properties of entropy: it is a concave and monotonically increasing function with respect to the number of equiprobable events. As a result, there always exists a maximum entropy value for a finite number of jobs within a hyper-period. Thus, Λ serves as the optimal point where the trade-off between task execution time and system entropy is balanced.

Let's explore the concept of MJTET. It refers to the minimum period of all jobs (task instances) where additional information cannot be introduced to the system. In other words, below this threshold, there is no new information being added to the system. Its bounds depend on applications; for example, for an embedded system, its limit is proportional to the inverse of maximum input/output signal/event bandwidth.

4.2 MIJET calculation: a methodology for task scheduling

The calculation of MIJET serves as a crucial step in establishing the lower limit for task duration and should be initiated at the beginning of the task scheduling process. To illustrate the applica-

tion of MIJET in the proposed scheduling method, let's first introduce the Nyquist-Shannon sampling theorem: "If we sample an analog signal uniformly at a rate that is greater than or equal to twice the signal's bandwidth, it becomes possible to perfectly recover the original analog signal from the discrete values" (Homayoon and Moon, 2020). The Nyquist-Shannon equation is given by:

$$f_s = 2 \times B, \quad (11)$$

where B represents the signal bandwidth, and f_s is the sampling frequency. Beyond this sampling rate, the likelihood of acquiring additional information from the environment is negligible.

In the context of modern industrial embedded systems, where analog signals are generated from PWM modulated signals, a correlation exists between PWM signal frequency (f_{PWM}) and signal sample rate (f_s):

$$u_{t_1} \times f_{\text{PWM}} = \alpha \times f_s = 2\alpha \times B. \quad (12)$$

Here, t_n denotes task n , u_{t_1} represents the utilization factor of Task1, associated with the highest frequency data sampling or signal generation, e.g. for computing the next PWM duty cycle. The constant α acts as a proportionality factor, enabling the construction of an analog filter with the fewest devices necessary to produce an acceptable output from a PWM signal. This is especially pertinent for tasks like generating three-phase inverter waveforms or controlling stereo audio signals. Experimentally it is sufficient to consider $\alpha = 1.5$ for many ordinary products, e.g. (Zirak and Roshani 2016).

To ensure a CPU generates a high-resolution PWM waveform with minimal jitter, it is essential to configure control registers to activate the option named the Phase-Frequency Correct procedure (Microchip Technology, 2023; Ünsalan et al.,

2023). With this operational procedure, during each PWM pulse cycle, the timer is required to count from 0 to the top value, such as 255, then decrement back to 0. The frequency of the PWM in phase-frequency correct mode ($f_{\text{PC,PWM}}$) is:

$$f_{\text{PC,PWM}} = \frac{f_{\text{clk}}}{2.N.TOP} = \frac{2\alpha \times B}{u_{t_1}}. \quad (13)$$

Here, N denotes the prescaler (e.g., =1), f_{clk} is the frequency of the CPU, and TOP is the top value of the corresponding CPU timer/counter. The maximum entropy of the PWM ramp signal, assuming it is assigned M bits, can be expressed as:

$$TOP = 2^M - 1. \quad (14)$$

To exemplify how this concept is used in the scheduling algorithm, let's consider various professional products implemented using this proposed platform and operating system (Khezerloo et al., 2017; Kiyani et al., 2011; Zirak et al., 2017; Zirak, 2023), and also see reference examples at www.xirac.com, www.asansor110.com and www.radaran.com. These products range from integrated elevator modules to power inverters, mass spectrometry systems, and smart homes, showcasing the versatility and robustness of the platform across diverse applications. Almost all products implemented with this operating system have a massive volume of application software and dozens of synchronized modules communicate with each other.

As per Eq. (13), consider a tiny 8-bit CPU with a clock frequency of $f_{\text{clk}} = 32$ MHz (or a thread with such a frequency which is generated inside another operating system), and PWM 8-bits registers ($TOP = 2^8 - 1 = 255$) in phase correct mode with $N=one$. This leads to a PWM frequency of:

$$f_{PC,PWM} = \frac{2\alpha \times B}{u_{t_1}} = \frac{32 \text{ MHz}}{2 \times 1 \times 255} = \frac{32 \text{ MHz}}{510} \approx 62.745 \text{ KHz}, \quad (15)$$

suitable for generating audio signals or driving switching mode power supplies or inverter carriers. With $\alpha = 1.5$ and $u_{t_1} = \frac{1}{2}$, the signal bandwidth is about $B \approx 10.46 \text{ KHz}$, providing ample capacity for various applications, such as generating audio signal or switching mode power supply or inverter carriers. With $u_{t_1} = \frac{1}{2}$, it implies that half of the time slots, i.e., task instances, are allocated to the periodic task with the highest frequency. Consequently, maximum entropy, and typically high-rate services such as the task request for real time pulse shaping, signal processing, or generally PWM duty cycle setting, are associated with Task 1 (τ_1). Conversely, the remaining half of the time slots is allocated to servicing the lower-frequency components, which inherently have lower entropy.

Viewing the MIJET as the basis of the PWM period, the quantity of instructions or CPU clock pulses carried out during one task period (I_{task}) can be expressed as:

$$I_{\text{task}} = \frac{f_{\text{clk}}}{f_{PC,\text{ramp}}} = 2 \cdot N \cdot \text{TOP} \quad (16)$$

About $2 \times 1 \times 255 = 510$ assembly instructions can be executed during each task period. However, due to the absence of dead-time, there is no limit to the number of instructions per task period. Moreover, the MIJET corresponds to 510 CPU clock pulses.

From this value of bandwidth (B) it can be concluded that, if we consider minimum informative job execution time (MIJET) to be a 510 CPU clock pulse (the denominator of the fraction of Eq. (8) then all types of analog audio signals and inverter waveforms up to 10.46 KHz bandwidth can be made with this PWM frequency with good

accuracy. On the other hand, MIJET is about 510 CPU assembly instructions if RISC technology is used, or;

$$\text{MIJET} = \text{WCET} = 510 \text{ CPU Clk} = \frac{510}{32 \times 10^6} \text{ s} \approx 16 \mu\text{s}. \quad (17)$$

4.3 Implementation of the task scheduler

In a specific example illustrating the use of hardware constraints to determine task repetition rates via maximum entropy and Lagrange multipliers, consider a processor with a 32 MHz clock speed and 8 KB of RAM. We examine three tasks: Task A with $\tau_A = 1500$ clock cycles and $r_A = 256$ bytes, Task B with $\tau_B = 2000$ clock cycles and $r_B = 128$ bytes, and Task C with $\tau_C = 1200$ clock and $r_C = 64$ bytes. The total constraints are a hyper-period duration $\mu = 5000$ clock cycles and RAM usage $R=384$ bytes.

By combining these constraints with the entropy function H using the Lagrange multiplier method (Eq. 2), and solving for p_i by setting partial derivatives to zero, we determine the task probabilities assuming $\lambda_1 = 0.001$ and $\lambda_2 = 0.01$: $p_A \approx 0.4$, $p_B \approx 0.3$ and $p_C \approx 0.3$.

This configuration ensures maximum entropy scheduling, optimizing task execution while respecting hardware limitations. By integrating these tasks directly into the kernel, the system achieves improved efficiency with minimal timing overhead and high resource utilization.

As a preliminary statement of the practical work conducted in alignment with the presented scenario, where limited prior statistical data, such as means values, are available, Eq. (8) yields the utilization factor for task i as $u_{t_i} = \frac{1}{2^i}$ where u_{t_i} represents the utilization factor, and $1 \leq i < \infty$. With these assumptions, the overall CPU utilization is given by $U = \sum_{i=1}^n u_{t_i} = \sum_{i=1}^n \left(\frac{1}{2}\right)^i \leq$

1. As the quantity of tasks (n) approaches infinity, the utilization converges towards unity. Consequently, the overall system becomes schedulable by maximum entropy, provided that the Nyquist-Shannon theorem or event rate sampling is adequately met, especially for higher repetition rate tasks. This sets the stage for the establishment of the task scheduling condition outlined in Corollary 1. In this context, Eq. (1) underscores that maximum entropy aligns with the tasks or events with the highest periodicity that are most probable. In the exemplified scenario, maximum entropy aligns with task τ_1 (Task 1), where $u_{\tau_1} = \frac{1}{2}$. For tasks with lower repetition rates, the probability of occurrence diminishes, reducing their impact on the total entropy or, in some cases, making their contribution negligible.

For constrained tasks scheduling, equation (7) demonstrates that entropy maximization is achieved through sequentially solving an optimization problem to determine the probability density of tasks and estimating stochastic request parameters to calculate the MIJET criteria. Moreover, maximizing system entropy and utilization requires integrating tasks from the application layer into the kernel, which entails a systematic organization of processes. This is achieved by mapping all processes and their corresponding driver architectures into appropriate kernel-level tasks, ensuring precision in alignment with the phase and frequency of their triggering events. Despite the numerous advantages, such as the ability to handle a multitude of processes and drivers concurrently within a modest 8-bit processor operating at 16 MHz or within another operating system, minimal context switches occur, using only a single timer (or thread) and freeing up the remaining hardware resources for user applications or the primary operating system.

In the versatile version of our proposed operating system, Task τ_1 , which operates at a repeti-

tion rate of about $\frac{f_{PC,PWM}}{2} \approx 31372$ Hz, is designated for processes with the highest repetition rate. These tasks typically include audio playing/recording/processing, inverter waveform generation/sampling, fast ADCs, multiplexing, data flash memory operations, and module interfacing.

Task τ_2 , operating at $\frac{f_{PC,PWM}}{4} \approx 15687$ Hz, is dedicated to processes like serial protocols for data transceiver (e.g., CANBUS, Zigbee, MODBUS, ISP with 38400 baud), DTMF passwords learning/decoding, graphical OLED/LCD display and remote control (ICP DAS USA Inc, 2024). Meanwhile, Task τ_3 , with a repetition rate of about $\frac{f_{PC,PWM}}{8} \approx 7843$ Hz, handles processes such as WIFI/GPRS/GSM/GPS module control, operations for boot loaders, and AGC (automatic gain control). Similarly, other tasks are allocated to other processes based on their repetition rates.

Using a simple overflow counter, task τ_4 (repetition rate of $\frac{f_{PC,PWM}}{16} \approx 3921$ Hz) is divided between three 1-KHz (period=1 ms for each) tasks with slight jitter and different phases. Every 1-KHz task encompasses processes associated with low rate I/O that involve analog/digital multiplexing features, scanning multiplexed displays, handling matrix I/O and keyboards, managing millisecond range operations, as well as identifying and controlling various standard peripherals. The hierarchical task structure allows the initiation of ten 100-Hz tasks, cascading into ten more 10-Hz tasks and finally generating ten 1-Hz tasks. For example, each 100-Hz task may encompass processes such as zone control, menu navigation, tele-control, and interface management, or it may trigger ten additional 10-Hz tasks. Similarly, ten 1-Hz tasks are generated in a similar manner.

Eq. (1) shows that lower repetition rates result in decreased probabilities of occurrence within a hyper-period, thereby reducing the overall entropy. Conversely, tasks with higher repetition

rates contribute more to the system's entropy. Therefore, these tasks need meticulous adjustment with a higher volume of information.

Considering the MIJET value specified in Eq. (17), it becomes apparent that the nominal duration of every task amounts to 510 CPU clock cycles. An important inference from Shannon's information theory is that the entropy of the aggregate of multiple random variables is less than the sum of their individual entropies (Zirak, 2023). Consequently, merging several analogous algorithms, i.e., several sub-tasks, as a larger task, instead of redundant analogous tasks, can be more efficient provided the length of the resultant task falls within the nominal task length range of 510 CPU clock pulses. Each task is conceptualized as a directed acyclic graph (DAG) consisting of a sequence of hierarchical sub-tasks.

The advantage of fixed priority scheduling methods, such as rate monotonic (RM) scheduling, is exemplified by their effective performance in overloaded states. In this method, less critical tasks miss deadlines instead of critical tasks, ensuring optimal operation. The proposed RTOS architecture leverages the advantages of RM even more effectively by implicitly aligning task importance with entropy, which is continuously expressed through the number of task repetitions. Unlike RM, the proposed method does not necessitate interruption of one task or sub-task by another, contributing to minimized interference, disturbance, and context switches. Given the non-critical nature of periodic tasks and the need to maximize entropy, the assumption that each task has an infinite deadline is practical. This is because if the execution time exceeds the deadline, the subsequent task experiences a delay in execution.

With a system carrying capacity of about $\frac{1}{\text{MIJET}} = \frac{32 \times 10^6}{510} = 62745$ tasks instances per hyper-period (e.g. one second), the probability

(Task_{*i,k*}) is 1/62745, where Task_{*i,k*} is the *k*th instance of the *i*th task. Using these calculated probabilities, the entropy value of Task_{*i,k*} is;

$$\begin{aligned} \text{Entropy} \\ (\text{Task}_{i,k}) &= P(\text{Task}_{i,k}) \log_2 \frac{1}{P(\text{Task}_{i,k})} = \\ &= \frac{1}{62745} \log_2 62745 = 2.5400 \text{ bit.} \end{aligned} \quad (18)$$

Therefore, the maximum entropy upper bound of the CPU within one hyper-period (hper) is:

$$\begin{aligned} \text{Entropy (CPU)} &= \sum_{i,k \in \text{hper}} \text{Entropy}(\text{Task}_{i,k}) = \\ &= \sum_{i=1}^{62745} \text{Entropy}(\text{Task}_{i,k}) = 15.9372 \text{ bits.} \end{aligned} \quad (19)$$

In this manner, a maximum entropy or maximum capacity limit, varying across different CPUs, is established in bits. In terms of load balancing, minimizing the entropy of overall tasks and maximizing the system's entropy outperforms scheduling based on the utilization factor. The suggested operating system scheduler extracts statistical traits from each task, aiming to minimize the entropy of whole tasks via task migration tactics.

Although we do not delve into this aspect here, the potential for further investigation into this topic is acknowledged.

4.4 Messaging system

In the proposed operating system, application modeling involves the use of DAGs, representing subsets of periodic or aperiodic tasks, each contributing to a well-defined function. With an infinite number of tasks, each DAG is inherently free-running, meaning it is idle and ready for a trigger to initiate its activity. The key innovation is that each free-running DAG, capable of emulating different and comprehensive systems, can be activated through control variables or a system of

messages. This flexibility, particularly as the number of DAGs approaches infinity, allows programming through various methods, such as setting control variables, utilizing DAG control through messaging methods via serial ports, or using local programming. A novel programming paradigm named "object emulated programming" is introduced, inheriting from object-oriented programming but with the distinctive feature that routines are activated and triggered implicitly by setting control variables.

In the design cycle of a robust product-oriented operating system, including the IOT, it is essential to consider that tasks of any application are distributed across several modules, making application scheduling a critical issue. Deployments, characteristics, and timing constraints of applications vary significantly. For example, a data acquisition application may take only 16 μ s to convert and record an analog signal to flash, while time-consuming applications, involving completion request/response and related processing between the main module and GPRS, inverter, or HVAC modules, may take several seconds or minutes.

To address these challenges, some systems use time-triggered communication protocols like TTCAN (time-triggered CAN), TTP (time-triggered protocol), and FlexRay (Jyotsna, 2024). Nevertheless, the complexity of TTP systems rises significantly to meet various functionalities within strict time constraints. These methods require considerable hardware, software, and bandwidth resources, making them less suitable for robust and diverse systems. Moreover, the unnecessary periodicity of other nodes within the application can lead to undue restrictions and compromise the overall scheduling capability of the system (Jyotsna, 2024, Zafari et al., 2019).

Driven by these difficulties, with the aim of augmenting the entropy of the entire system, we introduced an innovative messaging system to

schedule and synchronize tasks across applications. Every task within an application, executing on a particular module, is initiated by one or multiple messages originating from tasks running on a different module. This inter-modular message transaction occurs through various protocols via a common bus. Applications can be abstracted as DAGs, where the nodes correspond to messages or tasks, and the edges define the precedence constraints among the messages or tasks. The primary distinction is that tasks are defined by their execution time on a CPU, whereas messages are determined by their transmission time over the bus.

Building on the previously mentioned XIRAC method, which maximizes entropy for optimal scheduling of non-critical tasks in each module, the method is extended to schedule messages sent through a common bus or communication channel. An approach to achieve this goal is to use a single module, typically referred to as the main module, for scheduling all inter-module messages. The scheduling process is designed to maximize the feasible entropy of the communication system, while also ensuring that each periodic or aperiodic message adheres to its allowed entropy range.

In this scenario, certain OS tasks are associated with master-slave communication protocols such as MODBUS. These protocols involve request/response messages (frames) whose length is determined based on the specific requirements of the application. Similar to other tasks, each message, denoted as m_i , is associated with its own set of constraints, including a deadline (d_{m_i}) and a worst-case message transmission time ($WCTT_{m_i}$). Considering that the scheduler determines the length of every frame, the minimum reserved entropy, which applies to periodic messages and must be satisfied, can be expressed as follows:

$$H_{\min, \text{per}}(M) = E \left[\log_2 \left(\frac{d_{\max, m_i}}{WCTT_{m_i}} \right) \right] =$$

$$\sum_i \frac{WCTT_{m_i}}{d_{\max, m_i}} \log_2 \left(\frac{d_{\max, m_i}}{WCTT_{m_i}} \right) \text{ bits } \forall i \in \text{per}. \quad (20)$$

In this context, d_{\max, m_i} represents the maximum allowable deadline for message m_i that needs to be considered and *per* represents the set of periodic messages that should be repeated at least once within the desired period.

However, we use the following expression to estimate the maximum available entropy of the communication channel:

$$H_{\max, \text{bus}}(M) = E \left[\log_2 \left(\frac{P_{\text{hyper}, m_j}}{\min(WCTT_{m_j})} \right) \right] = \sum_j \frac{\min(WCTT_{m_j})}{P_{\text{hyper}, m_j}} \log_2 \left(\frac{P_{\text{hyper}, m_j}}{\min(WCTT_{m_j})} \right) \text{ bits } \forall j \in M \quad (21)$$

Ultimately, through the subtraction of the preceding two equations, we can derive the residual capacity of the communication channel, denoted as H_{rem} . This available capacity can be allocated for aperiodic messages or, in the absence of such messages, used for further repetitions of the periodic messages:

$$H_{\text{rem}}(M) = H_{\max, \text{bus}}(M) - H_{\min, \text{per}}(M). \quad (22)$$

For instance, in the design of an integrated elevator system product that is designed with this proposed RTOS, the control panel main module communicates with dozens of other modules (e.g., motor drive, encoder, car, and floor panels) using MODBUS-RTU protocol at 38400 bauds. For optimal movement, the main control module periodically reads one word of shaft encoder, drive and floor panels main parameters at least every 50 ms, 200 ms and 500 ms, respectively, with implicit deadlines, i.e. $d_{m_i} = p_{m_i}$. So, if the sizes of each send/receive frame, e.g. with function 3

(Read Holding Registers) is about 17 bytes then we have:

$$WCTT_{m_i} = \text{frame length in bytes} \times \frac{\text{bits of byte}}{\text{baud}} = 17 \times \frac{8}{38400} = 3.54 \text{ ms}, i = 1, 2, 3. \quad (23)$$

Thus, the minimum allocated entropy reserved for the rest of the messages is given by:

$$H_{\min, \text{per}}(M) = \frac{3.54}{50} \log_2 \left(\frac{50}{3.54} \right) + \frac{3.54}{200} \log_2 \left(\frac{200}{3.54} \right) + \frac{3.54}{500} \log_2 \left(\frac{500}{3.54} \right) = 0.2705 + 0.1030 + 0.0505 = 0.424 \text{ bits}. \quad (24)$$

Given the hyper-period of periodic messages is 1000 ms (the least common multiple of 50, 200, and 500 ms), the maximum available message entropy of the communication channel is:

$$H_{\max, \text{bus}}(M) \approx \log_2 \left(\frac{1000}{3.54} \right) = 8.1420 \text{ bits}. \quad (25)$$

In this example, the maximum available message entropy $H_{\max, \text{bus}}(M)$ is about 8.1420 bits, significantly greater than the minimum reserved entropy $H_{\min, \text{per}}(M)$ of about 0.424 bits. This suggests that the main module can schedule almost every aperiodic message, such as motor stop, set motor speed, and door open/close, among other periodic messages, ensuring that all deadlines are met. The proposed method for scheduling messages across communication channels can use different mechanisms, including polling-based algorithms or earliest deadline first (EDF). The primary programming paradigm for user application development is based on the polling-based method.

4.5 Object emulated programming for application

Considering the fundamental differences from common programming methods, where programmers typically start from scratch and gradually expand their programs using disparate and inconsistent libraries and threads, the proposed approach involves running the operating system as well as numerous libraries and drivers in the background of the processor's boot layer. This allows for minimal resource utilization while enabling unrestricted execution of foreground applications. As a result, application programmers can also write and execute their programs akin to bare-metal programming, using their preferred IDE and invoking maximum libraries or drivers leveraging control variable settings through pooling strategies. This flexibility extends to both local and remote environments, accommodating an infinite number of tasks and processes within the operating system. Within this framework, each resident task or process can be viewed as an active and tangible system implementation, introducing a new paradigm in application programming. In addition to inheriting properties from object-oriented programming, this approach empowers programmers to exert control over object implementations by adjusting the control variables. The term we use in this paper to describe this overarching structure is "object-emulated programming", which promises to offer a fresh perspective and transformative possibilities in the field of application programming.

In the operating system described, a single timer interrupt, occurring every 510 CPU clock pulses (approximately 16 μ s), is used for task scheduling. As a result, other processor resources are accessible and can be controlled by user applications or through tasks or another host and remote operating system. While the user application program is running in the foreground, the sched-

uler timer interrupt interrupts the program's execution when triggered. This interruption allows the execution of the current periodic task, provided it is ready to run. Once the periodic task is executed, control is then returned to the user application program. Consequently, the user application program can be perceived as a substantial preemptive task, while other periodic tasks within the operating system are considered non-preemptive.

The operating system is optimized to achieve maximum task entropy, with the aim of maximizing system utilization while keeping all system applications in a standby state. To activate system applications, data flags, referred to as control variables, are used, which are monitored by continuously polling their associated status indicators. These control variables can be set by the remote user application program through communication channels, such as function 16 of MODBUS, or through other local applications or tasks. This approach, termed runtime programming, provides a high degree of freedom for non-critical applications in terms of timing, enabling control of the operating system and programming under it without the need for a compiler, library, or threading. Additionally, there is a minimum requirement to call any functions since all of them are integrated in the kernel either in standby or active mode.

Below is a sample program in C# that exemplifies the "object emulated programming" paradigm, running on a computer but emulating a remote module with an address (Device Address) in near real time:

Program 1 A sample object-emulated program in C# on XIRAC operating system

```
1: My Computer. Communication Protocol="ModBus_TCP";
2: Device Zone device Zone=new Device Zone (Form1.
Device Address);
3: device Zone. On Off Password (zone Index, password);
...
```

```

4:if (device Zone. Remote Password [1]=="23E2A")
{
5: device Zone. Active Delay (3, 25);
6: device Zone. On Off State (3, true);
7: My Device. Display Buffer (device Dialler. Tel Number
(1),40);}
...

```

In this case, line 1 delineates the communication protocol used for data transmission with the remote module. Line 2 introduces a tangible object labeled as 'device Zone' representing the remote module, identified by the address 'Device Address', and equipped with various properties and methods. For instance, in line 3, the password of the zone with index 'zone Index' is set to 'password'. Line 4 waits for someone around the remote module to press the remote-control button corresponding to code '0x23E2A', and if detected, commands from lines 5 to 7 will be executed. These commands include setting the active delay of zone 3 to 25 s, turning on this zone, and displaying the phone number stored in memory index 2 on the connected screen of the remote module.

Following the development and manufacture of numerous functional products using this operating system, it has become evident that there's a pressing requirement for processor architectures and corresponding compilers that can be tailored to be compatible with the proposed operating system, thereby enhancing overall performance. Among other features, the program memory can be divided into intelligent blocks, capable of containing various tasks whose location, size and repetition rate are set by user applications or compiler, characterized with a specific allocation table and synchronized with the scheduler.

5 Performance assessment

In this section, we conduct a performance evaluation of the proposed operating system,

comparing it with other static and dynamic scheduling algorithms. Given that most comparison parameters, such as jitter, are a type of random signal and random variable, we used an experimental embedded system designed with the proposed platform as a test system. This is because simulator software is not suitable for comparison because it cannot emulate real systems from a dynamic perspective. This approach shifts the complexity from simulation to hardware resources.

Since the proposed framework can manage a substantial number of tasks and time slots, performance comparisons are typically carried out on a logarithmic scale. All graphs demonstrate the enhanced efficiency of the proposed approach as the number of tasks, processes, and program size experience significant growth. While it is important to note that the comparison between different methods might not always be conducted in precisely identical environments, the dynamic behavior comparison underscores the superiority of the proposed method as the user application leans towards a robust framework. These comparisons are conducted against scheduling algorithms such as rate monotonic (RM) and earliest deadline first (EDF).

In the first stage of the performance evaluation, Table 1 compares the total system entropy among the proposed method, EDF, and RM algorithms in relation to the number of task requests. Using Eq. (1), the total system entropy is calculated, wherein the probability $P(x_i)$ is determined as the quotient of executed instances of request i to the overall number of requests within a single hyper-period (one second).

The specific equation used to calculate total system entropy is not provided in the text but is referred to as Eq. (1). This equation likely involves a summation or integration over the set of non-repetitive user requests, capturing their probability distribution within the hyper-period. The

table visualizes how the total system entropy changes as the number of task requests varies. Entropy, in this context, is a measure of the system's disorder or unpredictability. Higher entropy indicates greater disorder, while lower entropy suggests a more predictable system.

The comparison in Table 1 provides insights into how well the proposed method handles variable numbers of task requests compared to EDF and RM algorithms. As per the table, once the transaction count exceeds 16384, the system entropy of the proposed method surpasses 0.8 bit. This suggests that the system's informational content approximately doubles compared to other algorithms. The table not only demonstrates the rise in system entropy but also facilitates a comparison of rejected task requests. This comparison provides insights into the system's robustness and its ability to manage a higher number of task requests effectively.

The subsequent inquiry will explore comprehending the influence of this heightened entropy on enhancing the performance of the proposed system. This could include examining how the system handles task scheduling, response times, and overall efficiency in the face of a more diverse and dynamic set of task requests.

Table 1 Total system entropy versus number of task requests for XIRAC, EDF and RM

Number of transactions	System entropy (bits)		
	XIRAC	EDF	RM
64	6	6	6
128	7	7	7
256	7.98	7.99	7.99
512	8.97	9.1	9.2
1024	9.9	9.78	9.71
2048	10.85	10.57	10.6
4096	11.93	11.45	11.34
8192	12.87	12.34	12.28
16 384	13.89	13.05	13.01

Fig. 1 depicts the delay in the user application loop, measured in microseconds, relative to the number of processes which are active. The graph offers insights into how the execution speed of the user application is impacted by the quantity of active processes and highlights the differences in performance between the proposed method and the EDF/RM algorithms. As the number of active processes increases, the proposed operating system demonstrates a significantly improved execution speed for the user application program. This implies that the system can efficiently handle a larger number of concurrent processes without a substantial increase in loop delay. The maximum allowable number of processes for EDF and RM is shown to be less than that of the proposed method. This suggests that the proposed method outperforms other popular algorithms, especially in scenarios with a high number of active processes. The ability to handle a greater number of processes is crucial for the efficiency of the operating system in large and complex applications. The scalability of the proposed method is highlighted, indicating its capability to maintain or even enhance performance as the system deals with a growing number of active processes. This scalability is a desirable trait in product-oriented real-time operating systems, especially in applications with varying workloads.

Fig. 2 shows the quantity correlation between tasks and active processes. This graph indicates that, as the processes within the user application experience a substantial increase, the count of active tasks within the proposed operating system maintains a comparatively stable trend when compared with systems reliant on EDF and RM algorithms.

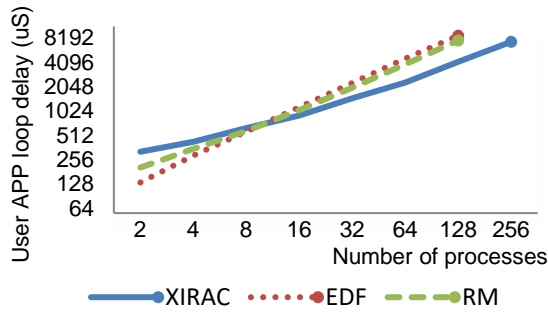


Fig. 1 User application loop delay versus number of active processes for the proposed method (XIRAC), EDF and RM algorithms

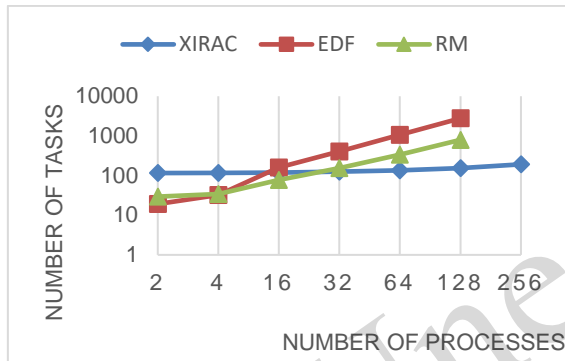


Fig. 2 Relationship between the number of tasks and the number of processes

In the proposed approach, a deliberate effort is made to sustain the count of active tasks at a consistently high level, ensuring optimal performance within a robust system architecture. The stability in the number of active tasks is a key feature that contributes to the proposed method's efficiency and reliability, especially in scenarios with varying workloads.

Note that, for the same reasons mentioned earlier, the maximum allowable number of processes for EDF and RM is observed to be less than that of the proposed method. This observation further emphasizes the scalability and adaptability of the proposed operating system, allowing it to handle a higher number of processes in robust product-oriented designs, without compro-

mising performance or system stability.

In Fig. 3, the success ratio (SR) is compared between the proposed method and the EDF algorithm, with respect to the number of inter-module transactions. The success ratio is a crucial metric, indicating the percentage of successfully scheduled tasks out of the total number of task arrivals. The equation for calculating the success ratio is given by (Avan et al., 2023):

$$SR = \frac{\text{Successfully scheduled tasks}}{\text{Total number of tasks arrival}} \quad (26)$$

As shown in this Figure, the success ratio in the proposed operating system exhibits a significant improvement as the number of transactions increases. The comparison with EDF-based methods highlights the superior performance of the proposed system, particularly in scenarios with a higher volume of inter-module transactions.

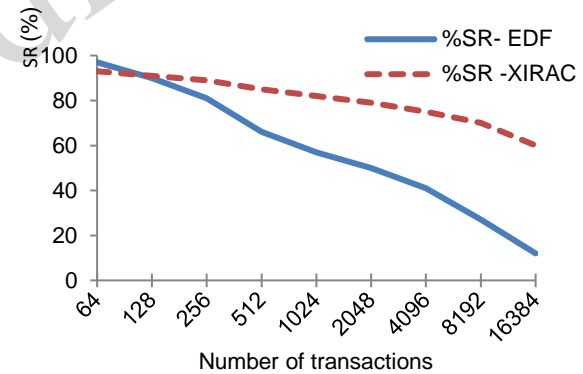


Fig. 3 Success ratio (SR) v. number of inter-module transactions

In another analysis, the distribution of task requests remains consistent with an average of 8192 requests per second. Different values of the parameter μ are applied to the probability distribution equation. Furthermore, a pioneering task scheduling framework is explored, incorporating a Poisson probability distribution characterized by varying task durations. In this Poisson model, the parameter λ is set equal to the mean number of

requests, with values of $\lambda=8192$ and $\lambda=4096$ being examined.

Fig. 4 illustrates the impact on the success ratio (SR) of using the maximum entropy probability distribution with a well-chosen parameter ($\mu=2$). The results confirm that incorporating this distribution significantly enhances the success ratio, emphasizing the effectiveness of the proposed method in achieving successful task scheduling under various conditions. In this experiment, we examined the effect of different μ values on task scheduling while keeping the task request distribution consistent, using the probability distribution described in Eq. (3). Additionally, we compared another task scheduling framework based on a Poisson probability distribution with variable task durations. In the Poisson model, the λ value was set to the average number of requests, specifically $\lambda=4096$ and $\lambda=8192$. Fig. 4 demonstrates that using the maximum entropy probability distribution with optimal parameters ($\mu=2$) significantly improves the success ratio (SR). This figure provides strong evidence that the proposed maximum entropy-based method considerably enhances the success and efficiency of the scheduling algorithm.

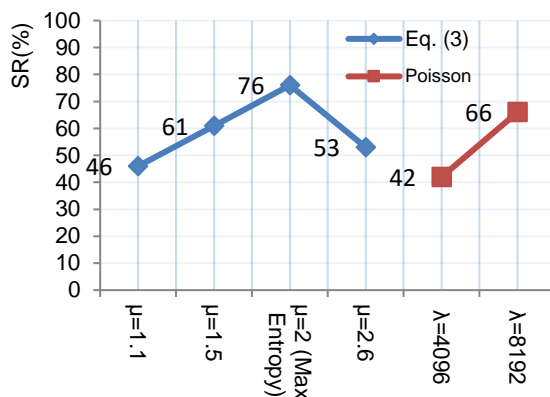


Fig. 4 Relationship between the success ratio (SR) and the probability distribution of tasks, with an average of 8192 requests per second

Fig. 5 illustrates the periodic jitter of a periodic task with a repetition rate of 1 KHz, as it relates to the number of processes that are active. We estimated the periodic jitter for each task using twice the standard deviation of its Gaussian function (Homayoon et al., 2020).

The graph demonstrates a notable enhancement in the jitter within our proposed operating system compared to systems using the EDF algorithm. This suggests that the proposed operating system exhibits superior performance in minimizing jitter and maintaining task regularity, particularly when faced with varying numbers of active processes.

It is crucial to highlight that the benefits of the proposed method become particularly noticeable when managing a small number of transactions. Yet, as the transaction volume increases, the jitter is prone to escalate in comparison to other popular methods. This observation highlights the trade-offs and considerations when scaling the system to handle a higher volume of transactions.

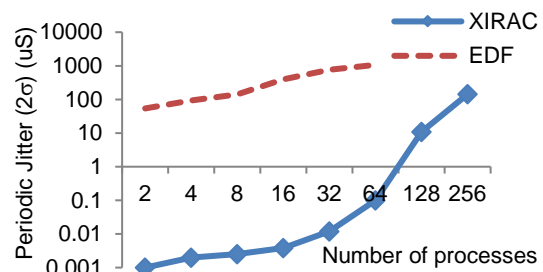


Fig. 5 Variation in jitter of a 1-KHz periodic task, in microseconds, as a function of the number of active processes

5 Discussion: Real-World Applications of XIRAC Operating System in Competitive Products

The XIRAC operating system demonstrates exceptional versatility and adaptability across a

variety of high-performance applications. Its ability to integrate seamlessly with different hardware platforms, from microcontrollers to complex electronic systems, highlights its potential as a unified solution for both advanced and consumer-grade products. For a brief introduction to the capabilities of the proposed operating system in its commercial form, see ‘Overview of the Xirac Operating System’ in the supplementary material.

Table 2 compares XIRAC, Arduino UNO and Rasp-berry Pi 3 boards. The first two structures are based on 8-bits, 20 MHz CPU, and 1 MHz SPI and I2C, and the corresponding values for the third structure are 64 bits, 1.5 GHz and 125 MHz. These structures serve as a basis for designing a sample product. The table indicates that XIRAC supports a significantly larger number of near-real-time applications compared to the libraries included in Arduino. To assess the impact of background applications on the execution of the main program, the state of one pin of the processor is inverted inside the main loop. For each state transition time of the output pulse, the average (mean delay) and jitter (double the standard deviation) are used to demonstrate the performance of XIRAC compared to bare-metal programming, i.e., without an operating system, with Arduino. Similarly, the same parameters were measured for the Raspberry Pi, which has a Debian-based operating system. Note that, despite the bootloader, XIRAC allows the use of valuable libraries of other platforms in its application layer. This flexibility provides developers with the option to leverage existing libraries and tools while benefiting from the enhanced capabilities of the XIRAC operating system.

Table 2 Comparison of XIRAC with Arduino UNO boards

	Libraries/ Applications	Mean delay	Jitter (2σ)	CPU active power
Ar-duino UNO	OLED+Flash+Keyboard+3* PWM output+MODBUS	850 us	24 us	0.43 W
Rasp-berry Pi 3	OLED+Flash+Keyboard+1* PWM output+MODBUS+Wifi about 25 other applications	47 us	49 us	4.2 W
XI-RAC	OLED+Flash+Keyboard+3-Acoustic out PWM+GSM+Wifi+MOD-BUS+Remote+Tele control+35 other applications	1.3 us	0.3 uS	0.45 W

According to Table 2, in the proposed method, due to the optimal partitioning of the maximum number of applications into tasks and the optimal scheduling of these tasks in the kernel, the latency and jitter in task execution and instruction processing are significantly lower compared to that in existing platforms that use a set of heterogeneous functions in their system design. Regarding CPU power consumption, as shown in the third column of Table 2, the use of XIRAC in product designs with considerable capabilities results in significantly lower power consumption compared to the Raspberry Pi 3, and it is within the range of Arduino UNO.

Fig. 6 provides an overview summarizing the three main components essential in each successful product-oriented project. These components are developed based on the proposed kernel platform, emphasizing robust applications. The three key sections encompass:

(i) Embedded Systems: engineered for unparallelled diversity and flexibility, offering support for infinite tasks and processes.

(ii) Diverse Communication Protocols: de-

signed to be compatible with different environments through the implementation of the proposed messaging system, enhancing inter-module communication efficiency.

(iii) Flexible Control Features: empowering local or remote system control based on the object emulated programming paradigm. This is achieved through easily developed, multiplatform interface software or other CPUs, facilitating seamless integration and control of the entire system.

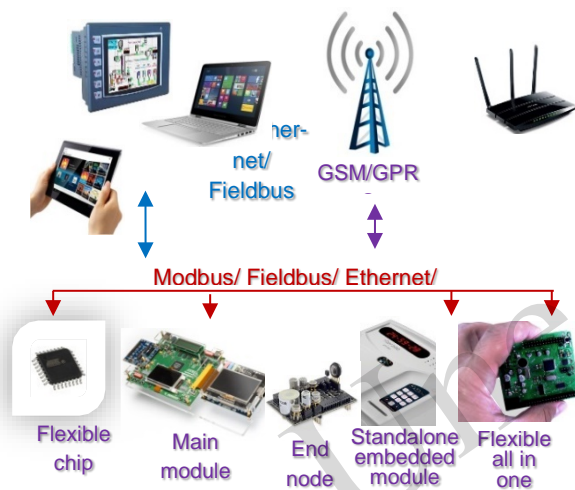


Fig. 6 Three main components required in each optimized product-oriented project, which are fully integrated into the proposed platform; (i) A collection of various modules, bottom of the figure (ii) inter-module communication protocols, represented by colored arrows, and (iii) flexible control features for interface software, Installed on a computer, tablet, mobile phone, or HMI.



Fig. 7 Elevator control system for gearless and hydraulic configurations, developed using XIRAC OS for precision task scheduling and safety-critical operations

Below are some notable real-world implementations showcasing the competitive advantages of XIRAC:

A. Elevator Integration System: Over the past six years, XIRAC has been integral to the development of gearless and hydraulic elevator systems, meeting stringent safety, reliability, and efficiency requirements (Fig. 7). Its real-time scheduling capabilities and deterministic behavior ensure precise motor control and fault-tolerant operation. These systems, now widely deployed, exemplify XIRAC's capability to manage inter-module communication in critical, high-demand applications.

B. Smart Home and Security Systems: With more than 1,000 units deployed, XIRAC serves as the backbone for smart home solutions, including security systems and energy management modules. This platform simplifies complex multi-node communication and data sharing across devices while ensuring low-latency responses to user inputs. Its scalability allows developers of varying expertise, including hobbyists, to implement and customize solutions effectively, fostering broader adoption in consumer markets (Fig. 8).



Fig. 8 Smart home control and security, for real-time event management and modular scalability

C. Mass Spectroscopy Systems: XIRAC has been utilized in the electronic subsystems of Paul traps and quadrupole mass spectrometers, managing RF, signaling, and data acquisition tasks (Fig 9). These applications benefit from XIRAC's ability to synchronize high-frequency data collection with real-time computational tasks. The operating system ensures precise timing and reliable communication, making it a critical component in high-precision scientific instrumentation.

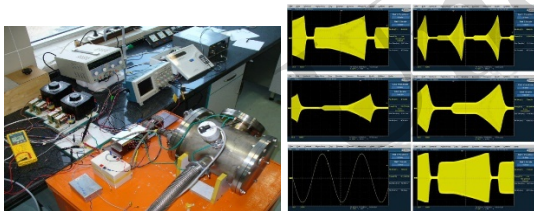


Fig. 9 Mass spectrometer electronics, enabling high-speed RF signaling and precise data acquisition for Paul traps and quadrupole systems.

D. Laser Power Supplies and Q-Switching: In the domain of laser technology, XIRAC supports Q-switching and synchronous power supplies for extended laser systems (Fig. 10). Its deterministic task scheduling ensures consistent laser pulse timing and power delivery, crucial for applications in research and industrial processing. The integration of XIRAC into these systems demonstrates its ability to handle low-level hardware interactions with high temporal precision.

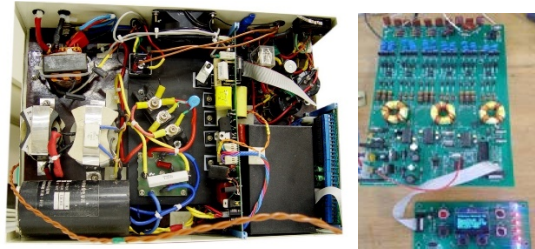


Fig. 10 Integrated Laser Q-switching (right) and synchronized power supply unit (left)

E. Standalone Chips and Modules for Development and Education: A noteworthy innovation is the delivery of XIRAC as a pre-installed operating system on standalone microcontroller chips. These chips cater to both professional developers and educational users, enabling them to design systems with enhanced portability and modularity. This feature facilitates rapid prototyping and product development without requiring a complete redesign for each platform.

6 Conclusions

In this study, we introduced an innovative approach that leverages Shannon's information theorem and the Nyquist-Shannon sampling theorem to redefine the landscape of operating systems, aiming at improved performance and versatility. The theoretical framework developed strategically combines these theorems to maximize system entropy tolerance and optimize task scheduling based on statistical data.

The culmination of our efforts is the creation of the XIRAC operating system, featuring a robust architectural core seamlessly integrated into the operating system kernel. This core provides a dynamic platform, empowering user application development with unparalleled flexibility. The introduction of data flags and the reduction of dependencies on compilers, libraries, and threading mark a departure from traditional program-

ming paradigms.

Our optimization approach, grounded in information theory, extends beyond conventional boundaries, sparking discussions about potential modifications to CPU architectures. XIRAC excels, particularly in scenarios with countless processes and tasks, showcasing its capability in handling robust and complex user application programs.

Additionally, the system's emphasis on maximizing feasible entropy in communication systems introduces novel methods for scheduling inter-module messages through common buses or communication channels. While contributing advancements to electronic product design, XIRAC is positioned as a contemporary solution for intricate systems, balancing efficiency with adaptability.

When combined with object-oriented programming, the integration of XIRAC reaches a new paradigm called object-emulated programming, enhancing the system's adaptability and efficiency through synergies between the robust architectural core and object-oriented principles.

In practical terms, our results demonstrate a substantial improvement in execution speed, the maximum allowable number of processes, success ratio, and jitter, particularly in resource-constrained scenarios. XIRAC emerges as a noteworthy addition to the operating system domain, reshaping competitive electronic product development by redefining the three pillars: an embedded system with maximal task diversity, communication protocols providing extensive flexibility, and intuitive interface software equipped with user-friendly and intuitive capabilities for development.

In summary, our research has not only introduced a novel operating system but also prompted a paradigm shift in the conceptualization and development of adaptable electronic systems, with an emphasis on balanced advancements and prac-

tical applicability.

Conflict of interest

The author declares that there is no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All data generated or analyzed during this study are included in this published article. Additional data related to this paper may be requested from the author.

References

- Tarasheh Karane Toos, (n.d.). XIRAC embedded platform and operating system. <https://www.xirac.com>. [Accessed: Jan 3, 2024].
- Asansore 110, (n.d.). Manufacturer of elevator products, <https://www.asansor110.com>. [Accessed: Dec 3, 2024].
- Abohamama AS, El-Ghamry A, Hamouda E, 2022. Real-time task scheduling algorithm for IoT-based applications in the cloud-fog environment. *J Netw Syst Manag*, 30(4):54. <https://doi.org/10.1007/s10922-022-09664-6>
- Avan A, Azim A, Mahmoud QH, 2023. A state-of-the-art review of task scheduling for edge computing: a delay-sensitive application perspective. *Electronics*, 12(12):2599. <https://doi.org/10.3390/electronics12122599>
- Fampa M, Lee J, 2024. An outer-approximation algorithm for maximum-entropy sampling. *Discrete Appl Math*, 347:271-284. <https://doi.org/10.1016/j.dam.2024.01.002>.
- Fang JH, Zhang R, Zhou AY, 2020. Load balance for distributed real-time computing systems. In: Zheng WA, Wang SP, Fang JH, et al. (Eds.), *East China Normal University Scientific Reports*. World Scientific, Singapore, pp. 83-107.
- Gaikwad JS, 2024. Multi-core system classification algorithms for scheduling in real-time systems. In: Kaiser MS, Xie JY, Rathore VS (Eds.), *ICT: Smart Systems and Technologies*. Springer, Singapore, p.229-237. https://doi.org/10.1007/978-981-99-9489-2_20
- He S, Li SZ, Chen Y, et al., 2015. Uncertainty analysis of race conditions in real-time systems. *Proc IEEE Int Conf on Software Quality, Reliability and Security*, p.227-232. <https://doi.org/10.1109/QRS.2015.41>

- ICP DAS USA Inc, 2024. Zigbee Wireless Modbus RTU Data Acquisition. https://www.icpdas-usa.com/zigbee_alliance_wireless_data_acquisition. [Accessed: Dec 30, 2024].
- Khezerloo D, Nedaie HA, Farhood B, et al., 2017. Optical computed tomography in PRESAGE® three-dimensional dosimetry: challenges and prospective. *J Cancer Res Ther*, 13(3):419-424. <https://doi.org/10.4103/0973-1482.202895>
- Khosravi Tanak A, Najafi M, Mohtashami Borzadaran GR, 2024. A new lifetime distribution by maximizing entropy: properties and applications. *Prob Eng Inf Sci*, 38(1):189-206. <https://doi.org/10.1017/S0269964823000062>
- Kiyani A, Abdollahzadeh M, Sadat Kiai SM, et al., 2011. Designing of a quadrupole Paul ion trap. *J Fusion Energy*, 30(4):291-293. <https://doi.org/10.1007/s10894-010-9369-9>
- Kondaveeti HK, Kumaravelu NK, Vanambathina SD, et al., 2021. A systematic literature review on prototyping with Arduino: applications, challenges, advantages, and limitations. *Comput Sci Rev*, 40:100364. <https://doi.org/10.1016/j.cosrev.2021.100364>
- Li SZ, Zhang YZ, Luo HY, et al., 2018. Race-condition-aware and hardware-oriented task partitioning and scheduling using entropy maximization. *IEEE Trans Parallel Distrib Syst*, 29(7):1589-1604. <https://doi.org/10.1109/TPDS.2017.2784829>
- Lisitsin DV, Gavrilov KV, 2024. The use of maximum entropy principle to construct robust estimators under point Bayesian contamination. Part I. *Appl Math Control Sci*, (1):55-72. <https://doi.org/10.15593/2499-9873/2024.1.04>
- Microchip Technology, n.d. Microchip Studio for AVR® and SAM Devices. Available at: <https://www.microchip.com/en-us/tools-resources/develop/microchip-studio>. [Accessed: Dec 5, 2024].
- Min-Allah N, Khan SU, Yongji W, 2012. Optimal task execution times for periodic tasks using nonlinear constrained optimization. *J Supercomput*, 59(3):1120-1138. <https://doi.org/10.1007/s11227-010-0506-z>
- Najmi HA, Moon TK, 2020. *Advanced Signal Processing: A Concise Guide*. McGraw-Hill, New York, USA.
- Rincón CA, Cheng AMK, 2017. Using information theory principles to schedule real-time tasks. *Proc 51st Annual Conf on Information Sciences and Systems*, p.1-6. <https://doi.org/10.1109/CISS.2017.7926091>
- Rincón CAC, Cheng AMK, 2018. SITSa-RT: an information theory inspired real-time multiprocessor scheduler. *Proc IEEE 21st Int Symp on Real-Time Distributed Computing*, p.156-163. <https://doi.org/10.1109/ISORC.2018.00032>
- Schäffler S, 2024. *Mathematics of Information: Theory and Applications of Shannon-Wiener Information*. Springer, Berlin, Heidelberg, Germany.
- Scharfenaker E, Yang J, 2020. Maximum entropy economics. *Eur Phys J Spec Top*, 229(9):1577-1590. <https://doi.org/10.1140/epjst/e2020-000029-4>
- Shannon CE, 1948. A mathematical theory of communication. *Bell Syst Tech J*, 27(3):379-423. <https://doi.org/10.1002/j.1538-7305.1948.tb01338.x>
- Sharma R, Nitin, 2013. Visualization of information theoretic maximum entropy model in real time distributed system. *Proc 3rd Int Conf on Advances in Computing and Communications*, p.282-286. <https://doi.org/10.1109/ICACC.2013.60>
- Ünsalan C, Gürhan HD, Yücel ME, 2022. *Embedded System Design with ARM Cortex-M Microcontrollers: Applications with C, C++ and MicroPython*. Springer, Cham, Switzerland. <https://doi.org/10.1007/978-3-030-88439-0>
- Wang KC, 2017. *Embedded and Real-Time Operating Systems*. Springer, Cham, Switzerland.
- Zafari A, Larsson E, Tillenius M, 2019. DuctTeip: an efficient programming model for distributed task-based parallel computing. *Parallel Comput*, 90:102582. <https://doi.org/10.1016/j.parco.2019.102582>
- Zirak A, 2023. XIRAC-Q: a near-real-time quantum operating system scheduling structure based on Shannon information theorem. *Quantum Inf Process*, 22(11):403. <https://doi.org/10.1007/s11128-023-04155-2>
- Zirak A, Beygi P, Mirzakhah S, 2017. Dynamic estimation of the modeling error statistics in Diffuse Optical Tomography. *Inverse Probl Sci Eng*, 25(4):492-505. <https://doi.org/10.1080/17415977.2016.1169280>
- Zirak AR, Roshani S, 2016. A reduced switch voltage stress class E power amplifier using harmonic control network. *Int J Adv Comput Sci Appl*, 7(5):38-42. <https://doi.org/10.14569/IJACSA.2016.070507>