International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 1 Volume No.8 January - February – 2024
DOI:10.46647/ijetms.2024.v08i01.005 ISSN: 2581-4621

Comparative Analysis of EmbeddedOperating Systems: A Criteria-Based Evaluation

Sanika More[1], Shivam Mukhede[2] , Mrs. Minal Deshmukh[3]
[1]*UG,Electronics and Telecommunication Engineering,BRACT's Vishwakarma Institute of Information Technology, Pune, India*
[2]*UG,Electronics and Telecommunication Engineering,BRACT's Vishwakarma Institute of Information Technology, Pune, India*
[3]*Associate Professor,Electronics and Telecommunication Engineering,BRACT's Vishwakarma Institute of Information Technology, Pune, India*
Corresponding Author Orcid ID: 0009-0007-9227-9940

ABSTRACT
Nowadays, the Internet of Things (IoT) encompasses a wide range of applications, including smart home systems, healthcare devices, smart parking systems, and smart transportation solutions. For these types of IoT applications, embedded operating systems play a crucial role by providing excellent real-time performance and reliability. The important purpose of this system is to efficiently perform specific tasks. This paper provides a quantitative and qualitative analysis of embedded operating systems. The study in this paper focused on a selection of systems commonly utilized in industrial and academic environments. We evaluated embedded operating systems like Linux, Android, QNX, FreeRTOS, TinyOS, PalmOS, Ubuntu, LiteOS, VxWorks, and Integrity based on specific criteria. It will help industries and academics select embedded operating systems according to requirements for performing specific tasks.
Keywords—RTOS, Embedded systems, Internet of Things, Software, Multi-core

1. Introduction

Embedded operating systems consist of both hardware and software, which means they contain software and a processor. They are self-contained in the device and reside in the ROM. Typically suitable for specific functions or use cases rather than applications or tasks. Embedded systems have a small amount of memory and processing power. Embedded OS runs on computers that control devices like microwave ovens, TV sets, and mobile telephones. They are similar to real-time operating systems (RTOS) but are constrained by limitations in terms of size, power, and memory.
In the early 20th century, mechanical and electrical systems were embedded in several devices, such as early calculators and industrial machinery. The Apollo Guidance Computer (AGC) marked the introduction of embedded operating systems in the 1960s, representing a significant historical milestone in the development of embedded systems. It played a crucial role in NASA's Apollo spacecraft, delivering navigation, control capabilities, and guidance during historic lunar missions. The 1970s and 1980s witnessed the emergence of microcontrollers and microprocessors, enabling more sophisticated embedded systems. In the 1980s and 1990s, real-time operating systems were established. They provide deterministic response times and are used in many industries, like aerospace and telecommunications. In the late 1990s, embedded Linux was developed, which is a more flexible and open-source alternative to several types of applications. The early 2000s witnessed the emergence of smartphones with Android and iOS operating systems that support embedded functions. In recent years, the use of the Internet of Things has increased, where embedded operating systems play a significant role. Many companies now develop customized embedded operating systems for specific devices and applications.

DOI:10.46647/ijetms.2024.v08i01.005 ISSN: 2581-4621

Embedded operating systems are utilized in a wide range of applications where dedicated, specialized software is required to control and manage hardware. In smart TVs, embedded OSs power the user interfaces, networking, and functionality of modern smart televisions. They are used in PLCs (programmable logic controllers) to control machinery and manufacturing processes in industries. In patient monitoring equipment, embedded OSs are used in many devices, like ECG monitors and infusion pumps. It is also used in communication systems for routing and switching. Smartphones run on embedded OSs such as Android and iOS. In-home automation embedded systems are used in many pieces of equipment, like microwave ovens, washing machines, dishwashers, and so on. Wearable devices like smartwatches and fitness trackers use embedded OS for functionality. Devices that act as intermediaries between IoT sensors and the cloud often run embedded OSs to manage data transmission and processing. So, these are the few applications of embedded operating systems, showcasing their versatility and importance in various industries and technologies.

2. Literature review

Yew Ho Hee et al., “Embedded operating systems and industrial applications: a review." This review provides a systematic examination of the shared characteristics and distinctions between different embedded operating system solutions. It also analyzes the factors that play a role in the decision-making process when choosing the most appropriate solution for specific applications. The review discusses three standard solutions, including real-time operating systems (RTOS), super loops, and cooperative systems. By categorizing tasks into foreground and background execution domains, the paper explores the core principles and operational concepts associated with each of these solutions.

Karunakar Pothuganti et al., "A Comparative Study of Real-Time Operating Systems for Embedded Systems" This paper presents a comprehensive analysis encompassing both quantitative and qualitative findings obtained from the evaluation of real-time operating systems (RTOS). The study encompassed the examination of multiple systems, such as Windows CE, VxWorks, QNX Neutrino, Linux, and RTAI-Linux, which are widely used in both industrial and academic contexts. Furthermore, the analysis incorporates Windows XP as a reference point for traditional non-real-time operating systems, as these are often used inadvertently in instrumentation and control applications. The paper also covers aspects such as worst-case response times, latency, clock accuracy, and response time.

Luis Fernando Friedrich et al., “A Review of Operating System Infrastructure for Real-Time Embedded Software” This paper offers an overview of the strategies and methodologies associated with operating systems used to address the changing requirements in the field. Currently, there is a growing trend towards implementing embedded systems in a distributed fashion, and a wide spectrum of high-performance distributed embedded systems have been conceptualized and deployed. The design of embedded systems heavily relies on the efficient interaction of distributed processors, and it is clear that due consideration must be given to the software infrastructure, including operating systems, to ensure they can provide the necessary functionality to meet these evolving demands.

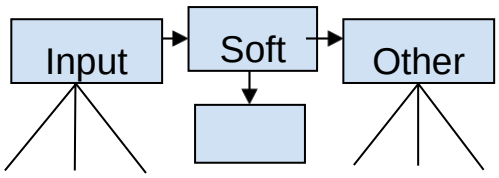
Robert P. Dick et al., "Power Analysis of Embedded Operating Systems" In this research paper, the authors delve into the power consumption analysis of real-time operating systems (RTOS), a critical component within the system software layer. Despite the widespread adoption and significant role of RTOS in mobile and low-power embedded systems, limited knowledge exists regarding their power consumption characteristics. Their studies offer power consumption profiles for commercial real-time operating systems, specifically µC/OS when executed on an embedded system based on the Fujitsu SPARClite processor. The authors demonstrate various strategies for designing application software that effectively leverages the RTOS to achieve energy efficiency. This research serves as an initial step in establishing a structured approach to RTOS power modeling and optimisation.

Sagar PM, "Embedded Operating Systems for Real-Time Applications" In this paper, the authors provide information about the workings of embedded systems, what operating systems mean, the

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 1 Volume No.8 January - February – 2024
DOI:10.46647/ijetms.2024.v08i01.005 ISSN: 2581-4621

features of real-time operating systems, and also about case studies on different real-time embedded operating systems. Real-time operating systems facilitate the development and scalability of real-time applications and help to form processes by segregating application code into distinct tasks. Additionally, an RTOS optimizes system resources and provides valuable services, including mailboxes, time delays, time-outs, and more.

3. Embedded Operating System Parts And Operation



Mouse Keyboard Switches Printers Monitors Buzzers
Fig 1. Process of Embedded Operating System

It is a computer operating system designed for use in embedded computer systems. These operating systems are manufactured to be compact, efficient, resourceful, and reliable, and to eliminate features unnecessary for specialized applications. It gives the output to humans in forms like audio, text messages, images, and video. Mainly, we use software in embedded operating systems for programming the code. This programming code helps to convert machine-level languages into user-understandable programming languages such as Java, C, and C++.
Input Tool: It is used to transmit data from users to the system. Users serve as the source of input. Keyboard, mouse, microphone, switches, sensors, etc. come under the input tools.
Output Tools: These devices are used to receive signals from software. They present results in the form of text, sounds, images, or videos. Some common devices are printers, monitors, LED displays, LCD screens, various types of motors, buzzers, and so on.
Memory: The memory is used to store data. Some of the memory devices are SD cards, RAM, ROM, EEPROM (Electrically Erasable Programmable Read-Only Memory), and Flash memory. The memory devices used in the embedded system are non-volatile RAM, volatile RAM, dynamic random-access memory, and so on.
Software: Software has supreme importance within embedded operating systems as it acts as the intermediary between input and output tools. It assumes the vital responsibility of controlling and coordinating the hardware components of an embedded system, ensuring the efficient and dependable execution of specific tasks.

4. Methodology

Now we are going to see information about various embedded operating systems based on their specialty, scheduling, priority level, and applications based on their performance.
Linux on embedded systems
Linux on embedded systems refers to the use of the Linux operating system on embedded computing devices. This system is designed for specific tasks such as controlling machinery appliances, monitoring sensors, or running in-car entertainment systems. Linux is open-source software. There are real-time Linux variants like PREEMPT-RT that provide predictable and low-latency responses. The compatibility of this embedded OS is versatile but can be resource-intensive. Security depends on configuration and additional tools. Linux uses a priority-based scheduling algorithm. The Linux scheduler is a Completely Fair Scheduler (CFS) for process management. Linux supports real-time scheduling with SCHED_FIFO and SCHED_RR policies, making it suitable for real-time applications. Linux offers control groups for managing and monitoring resource allocation, allowing

DOI:10.46647/ijetms.2024.v08i01.005 ISSN: 2581-4621

fine-grained control of scheduling and resource usage. Linux allows setting CPU affinity for processes, ensuring that they run on specific CPU cores. The Linux scheduler includes load-balancing mechanisms to distribute tasks evenly across CPU cores.

Android

Android is an open-source operating system. It is designed for mobile devices but is also used in embedded systems. Apps and development resources for Android have a vast ecosystem. Android uses the Linux kernel for core operations, and it adds its own application framework for managing app processes and scheduling. Android uses the Zygote process to create new app processes efficiently, reducing startup times. Android can be resource-intensive, making it suitable for more powerful embedded devices. It provides the Looper class for managing message loops and event scheduling in apps. It has a user-friendly interface with extensive customization options. Security mechanisms are provided by Android. It is essential for mobile and IOT security.

QNX

QNX is a real-time operating system with a microkernel architecture. QNX is known for its reliability and is often used in safety-critical and automotive applications. It offers deterministic real-time performance. In the automotive industry, QNX is popular for in-vehicle infotainment and control systems. QNX can scale from small systems to complex embedded applications. QNX security features are used in applications where security is paramount. QNX's process scheduling is based on priorities, deadlines, and resource constraints, making it suitable for critical real-time applications. QNX provides POSIX-compliant (portable operating system interface for Unix) APIs, allowing developers to use familiar scheduling mechanisms. QNX offers resource managers the ability to control and allocate system resources efficiently, enhancing scheduling predictability.

FreeRTOS

The FreeRTOS operating system is designed for real-time applications with predictable scheduling. FreeRTOS uses a priority-based scheduling algorithm. FreeRTOS has a small memory and storage footprint, making it suitable for resource-constrained devices. It is highly portable. It can be used for hardware platforms. FreeRTOS employs a task-based programming model for efficiency and simplicity. There is an active community providing support and libraries for FreeRTOS. FreeRTOS supports preemptive scheduling, ensuring that higher-priority tasks can interrupt lower-priority ones. It provides round-robin scheduling for tasks with the same priority, enhancing fairness. FreeRTOS tasks can be delayed for a specified time or until an event occurs.

TinyOS

TinyOS is designed for low-power wireless sensor network applications. TinyOS follows an event-driven programming model, where tasks are scheduled based on events. TinyOS minimizes resource consumption to extend battery life. It is designed for applications with intermittent data. Applications are built using a component-based architecture for reusability. TinyOS offers a simple and efficient framework for sensor data processing. TinyOS is made based on the priority of events, and higher-priority events preempt lower-priority ones. TinyOS is optimized for low-power operation, with scheduling mechanisms designed to conserve energy.

PalmOS

PalmOS was designed for stylus-based touchscreen devices, making it intuitive for its time. PalmOS was a single-tasking operating system. It could only run one application at a time. PalmOS was widely used in Palm pilot devices for personal organization and data management. It supported a range of third-party applications, enhancing its functionality. Applications in PalmOS use cooperative multitasking, where they yield control to other applications voluntarily. PalmOS devices had limited memory and storage capacity. Memory constraints and management affect scheduling in PalmOS, as freeing memory is essential to accommodating new tasks. It supported data synchronization with desktop computers for data backup and transfer. It lacked support for multimedia and advanced internet connectivity. Event-driven applications in PalmOS are scheduled to handle user and system events. PalmOS manages foreground and background tasks, with priority given to foreground applications.

DOI:10.46647/ijetms.2024.v08i01.005 ISSN: 2581-4621

Ubuntu

Ubuntu is a popular Linux distribution known for its user-friendly interface and extensive software repository. Ubuntu offers separate distributions for desktop and server use, providing a wide range of applications and services. Ubuntu is open-source, which allows for customization and community-driven development. This software center simplifies installation and management. Ubuntu can be configured with real-time extensions like the PREEMPT-RT patch set to support real-time tasks. It supports a broad range of hardware, including x86 and ARM architectures. Ubuntu implements nice values to prioritize processes, allowing users to influence scheduling. Priority levels can be assigned to processes to influence their scheduling order.

Huawei LiteOS

Huawei LiteOS was an open-source, lightweight real-time operating system. It has a small kernel, less than 10 kilobytes. LiteOS is energy-efficient and has a fast startup within milliseconds. It supports Wi-Fi, Zigbee, Ethernet, and different IoT protocols. It is designed for low-power, resource-constrained IoT devices and offers efficient performance. LiteOS includes a hierarchical file system and a wireless shell interface for user interaction, which operate through UNIX-like commands. It provides kernel support for dynamic loading and native execution of multithreaded applications. It facilitates software updates through a clear separation between kernel and user applications, connected via a range of system calls. LiteOS supports round-robin, fixed priority, earliest deadline first, and deadline-based scheduling.

VxWorks

VxWorks is renowned for its unparalleled deterministic performance. It is designed for a scalable, safe, secure, and reliable operating environment ideal for mission-critical computing systems with the highest demands. It is often used in specialized applications with less focus on user interfaces. The VxWorks development environment contains the kernel, other software and hardware technologies, and board support packages. Scalability, safety, security, graphics, and connection have all been enhanced to meet the demands of the Internet of Things (IoT). VxWorks performs functions of task management, memory management, and interrupts. There is no process for context switching. The interrupt vector table stores the OSR address, which is supplied directly from the hardware. It supports round-robin and priority scheduling.

Integrity

Integrity is an RTOS known for its real-time capabilities and hardware compatibility. It has WPA2, Bluetooth, and 3G Wi-Fi support. For routing, it uses the IPv4 and IPv6 protocols. Integrity supports 2D, 3D, and OpenGL graphics. It is a safe and secure separation microkernel architecture. The integrated architecture supports multiple protected virtual address spaces, each of which can restrain multiple application tasks. Mostly, this embedded OS is used in safety-critical systems with budget considerations. It supports round-robin, priority-based, fixed-priority preemptive, and deadline-based scheduling. Integrity supports Rate Monotonic Scheduling (RMS), a well-known real-time scheduling algorithm that assigns priorities based on task periods; shorter periods receive higher priorities.

5. Results And Discussion

Selecting the best embedded operating system is a complex and critical task. The selection process depends on the specific application's requirements. Making the right choice can lead to a cost-effective solution and the ability to achieve excellent results within project deadlines. With the increasing use of the Internet of Things (IoT), the demand for embedded OS is increasing.

Here we provide information about the type of kernel, threading, scheduling, and priority level in different embedded operating systems.

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 1 Volume No.8 January - February – 2024
DOI:10.46647/ijetms.2024.v08i01.005 ISSN: 2581-4621

Criteria	Linux	Android	QNX	FreeRTOS	TinyOS	PalmOS	Ubuntu	LiteOS	VxWorks	Integrity
Kernel	Linux Kernel	Linux long-term supported (LTS) kernel	Microkernel	Real-Time Microkernel	-	Monolithic	(Monolithic)Linux Kernel	Real-Time	Monolithic	Microkernel
Threads	Both User and Kernel Level	Both User and Kernel Level	Kernel Level	Kernel Level	User Level	User Level	Kernel Level	User Level	Kernel Level	Kernel Level
Scheduling	Priority based	Priority based	Priority based	Preemptive	Priority based	Priority based	Priority based	Round-robin, fixed priority	Round-robin and priority	Round-robin, priority based
Priority Level	40 Level	40 Level	256 Level	256 Level	32 or 16 Levels	32 Levels	40 Level	32 Level	256 Level	256 Level

Table 1: Segregation of different embedded operating systems according process specification

By studying different types of embedded OS with the help of criteria like application requirements, resource efficiency, safety and reliability, etc. we provide a table. This table shows the segregation of different embedded operating systems according to the given criteria. We showed which criteria are available in which embedded operating system in yes and no format.

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 1 Volume No.8 January - February – 2024
DOI:10.46647/ijetms.2024.v08i01.005 ISSN: 2581-4621

Criteria	Linux	Android	QNX	FreeRTOS	TinyOS	PalmOS	Ubuntu	LiteOS	VxWorks	Integrity
Application Requirements	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Real-Time Capabilities	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes
Resource efficiency	No	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes
Safety and Reliability	No	No	No	Yes	No	No	No	No	Yes	Yes
Scalability	Yes	Yes	No	No	No	No	Yes	Yes	No	No
Hardware Compatibility	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Use Case Alignment	No	Yes	No	No	Yes	Yes	No	Yes	No	No
Performance Testing	No	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes

Table 2: Segregation of different embedded operating systems based on criteria

DOI:10.46647/ijetms.2024.v08i01.005 ISSN: 2581-4621

The choice of an operating system among the 10 listed depends on the specific requirements of our projects. Specifically, select the real-time embedded operating system because it can perform tasks in any environment.

CONCLUSION

Linux emerges as a highly versatile operating system, well-rounded option for making it an excellent choice for a wide range of applications. Android is ideal for mobile devices, while QNX, VxWorks and Integrity excel in real-time and safety-critical applications. FreeRTOS is lightweight and suitable for resource-constrained systems, while Ubuntu is a general-purpose option aligned with many criteria. Ultimately, the decision should be guided by our project's unique needs, considering factors such as real-time capabilities, resource efficiency, hardware compatiblity, security etc.

Our comprehensive study reveals that embedded operating systems are like tools, and each designed is for specific jobs. Imagine it's like having a toolbox with different tools for different tasks. In many cases, especially in complex systems, we use more than one tool(or embedded operating system) to get everything done efficiently. This allows each part of the system to work at its best, making the whole thing work better.

References

1. Ywe Ho Hee, Mohamad Tarmizi Abu Seman, Mohamad Khairi Ishak, Mohd Shahrimie Mohd Assari "Embedded operating system and industrial applications: a review" Bulletin of Electrical Engineering and Informatics,Vol. 10,No.3,pp.1687-1700,June 2021.
2. Karunakar Pothuganti, Swathi Pothuganti, Aredo Haile "A Comparative Study of Real Time OPerating Systems for Embedded Systems" International Journal of Innovative Research in Computer and Communication Engineering, Vol. 4, Issue 6, pp. 12008-12013, June 2016.
3. Aravinda Prabhu. S, Ganesh Prabhu, Preethika R "A Study of Operating System for Embedded Systems" International Journal of Latest Trends in Engineering and Technology, pp. 54-58, 2016
4.Luis Fernando Friedrich, Mario A. R. Dantas "A Review of Operating System Infrastructure for Real-Time Embedded Software", Journal of Communication and Computer 12, pp.273-285, 2015
5.Sachin R. Sakhare, Dr. M.S. Ali "An Adaptive Framework for the Selection of Embedded Operating Systems" , International Journal of Scientific and Engineering Research, Vol.2, Issue 8,Aug 2011.
6. Sagar P M "Embedded Operating Systems for Real-Time Applications", M.tech credit seminar report, IIT Bombay, Nov 2002.
7. Robert P. Dick, Anand Raghunathan, Ganesh Lakshminarayana, Niraj K. Jha "Power Analysis of Embedded Operating Systems",37th Design Automation Conference, pp. 312-315,June 2000