Unit 3
Real-Time Operating System (RTOS)-I

Structure of the Unit

3.1 Unit Outcomes

After the successful completion of this unit, the student will be able to:

1. Define Real time operating system
2. Identify the characteristics of real time operating system.
3. Analyze the RTOS types.

3.2 Background

A real-time system is a type of computer system that not only demands accurate computing results but also necessitates that these results are generated within a predefined timeframe or deadline. Even if the results are correct, if they are produced after the specified deadline has lapsed, they might hold little to no practical value. To better understand this concept, let's consider the example of an autonomous robot responsible for delivering mail within an office building. If the robot's visual control system correctly identifies a wall, but this identification occurs after the robot has already collided with the wall, the system's performance falls short of meeting its intended purpose.

On the other hand, requirement of timing is much lenient for normal desktop systems. Take, for instance, an interactive desktop computer system. While it is advantageous for such a system to promptly respond to user interactions, it is not an absolute necessity. On the other end of the spectrum, there are systems like batch-processing setups that may not have any specific timing prerequisites whatsoever. In these cases, tasks are executed in batches, and the system isn't under pressure to meet stringent timing constraints.

Real-time systems that operate on regular computer hardware find application in a wide number of fields.

Additionally, many real-time systems are integrated into "specialized devices," which can range from everyday household items like microwave ovens and dishwashers to popular consumer electronics like cameras and MP3 players, as well as communication gadgets like cell phones and Blackberry handheld devices. An embedded system is a type of computing device that is a component of a larger system.

In other words, real-time systems aren't limited to just regular computers – they're everywhere. Think about your microwave or your camera – they have small computer-like parts that help them work precisely and on time. These real-time systems are even inside bigger things like cars and planes, making sure everything runs smoothly. When we say "embedded system," we mean that these computer parts are tucked inside larger things. Sometimes, you might not even realize there's a computer inside because it's quietly doing its job to make things work correctly. So, the world of real-time systems is bigger and more pervasive than we might think!

Certain real-time systems are categorized as safety-critical systems. In a safety-critical setup, when things go wrong – often because a task wasn't completed on time – it can lead to serious accidents or disasters. For instance, safety-critical systems encompass areas like military weapons systems, the braking systems that prevent skidding in cars, flight management systems in airplanes, and even health-related embedded systems such as pacemakers. In these situations, the real-time system has to react to events within specific time limits; if it doesn't, it could result in severe harm or even loss of life. However, most embedded systems aren't as critical in this way. Think of things like fax machines, microwave ovens, wristwatches, and networking devices like switches and routers. If these systems miss their timing targets, the worst consequence might be an unhappy user.

Real-time computing falls into two main types: hard and soft. A hard real-time system is the most strict; it ensures that super important tasks are completed within their deadlines, no exceptions. Safety-critical systems usually fall into this category. On the other hand, a soft real-time system is a bit more lenient. It just guarantees that an important task gets to go first and keeps its "first-in-line" spot until it's done. Many everyday operating systems, including Linux, fall into the soft real-time category, meaning they try to prioritize important tasks but aren't as strict as hard real-time systems in sticking to exact deadlines.

Real-time operating systems were developed for real time systems. Real-time operating systems offer essential functions like scheduling, resource management, synchronization, communication, precise timing, and input/output operations. They've come a long way from being specialized systems designed for specific tasks to becoming more versatile and general-purpose. For example, there are real-time versions of common operating systems like Linux.

The evolution of real-time operating systems has been quite interesting. Initially, they were completely predictable and focused on supporting critical tasks where safety was paramount. Over time, they've adapted to also cater to softer real-time applications, which are a bit more flexible with their timing. In the context of open real-time systems, there's a concept called "quality of service." This is often used in applications like multimedia and in complex distributed real-time systems. The goal is to ensure that these systems provide a certain level of performance and responsiveness, aligning with user expectations.

Researchers in the field of real-time systems have come up with new ideas and ways of thinking that have improved traditional operating systems. Some of these ideas have found their way into common operating systems that we use daily. Additionally, many unique ideas have contributed to the diverse range of operating systems available in the market today.

The real-time operating system (RTOS) market is quite diverse. It includes proprietary kernels, kernels

built using composition-based approaches, and real-time versions of well-known operating systems like Linux and Windows NT. This variety allows developers and users to choose the best-suited option for their specific real-time requirements.

3.3 Characteristics of Real-Time Operating Systems

Real-time operating systems exhibit distinct requirements in five key domains:

1. **Determinism**: This attribute pertains to an operating system's capability to execute operations at predetermined intervals or fixed times. While no system can be entirely deterministic when multiple processes compete for resources and processor time, a real-time operating system responds to service requests based on external events and predefined timings. The system's ability to deterministically fulfill these requests is based on two factors: its rapid response to interrupts and whether it possesses the capacity to manage all requests within stipulated timeframes. A useful metric to gauge deterministic functionality is the maximum delay starting from the arrival of a high-priority device interrupt to the commencement of servicing. In traditional non-real-time operating systems, this delay might span tens to hundreds of milliseconds, while in real-time operating systems, it could be constrained anywhere from a few microseconds to a millisecond.

2. **Responsiveness**: In contrast to determinism, responsiveness concentrates on the interval an operating system delays after acknowledging an interrupt. It delves into the duration it takes for the operating system to handle the interrupt after its acknowledgment. Aspects encompassed within responsiveness are:
   - The time required to initiate interrupt handling and initiate the execution of the interrupt service routine (ISR). If the ISR necessitates a context switch, the delay will be lengthier compared to when the ISR can be executed within the ongoing process's context.
   - The duration essential to perform the ISR, which generally depends on the underlying hardware platform.
   - The impact of interrupt nesting. If an ISR can be interrupted by the emergence of another interrupt, the servicing sequence will be postponed.

   Determinism and responsiveness combinedly establishe the response time to external events. Response time is of paramount importance for real-time systems as these systems must adhere to timing criteria imposed by individuals, devices and so on.

3. **User control** : In comparison to ordinary operating systems, real-time OSs offer a broader scope of user control. In conventional non-real-time systems, users either lack influence over the scheduling operations of the OS or possess limited options like categorizing users into priority classes. However, in a real-time setup, it becomes imperative to grant users intricate control over task priorities. This entails the ability to differentiate between hard and soft tasks, specifying relative priorities within each class. A real-time OS may even enable users to define details such as the utilization of paging or process swapping, the permanence of certain processes in main memory, the choice of disk transfer algorithms, rights allocated to processes across various priority bands, and more.

4. **Reliability** :Reliability assumes even greater significance in real-time systems compared to their non-real-time counterparts. In a non-real-time system, a transient malfunction might be resolved through a system reboot, while in a multiprocessor non-real-time environment, a processor failure might lead to diminished service until the faulty processor is fixed or replaced. Conversely, a real-time system operates in sync with and manages events in actual time. Any loss or degradation of performance can result in dire consequences, spanning financial losses, significant equipment damage, and even loss of life.

5. **Fail-Soft operation**:As with other aspects, the distinction between real-time and non-real-time operating systems lies on a spectrum. Even within a real-time system, provisions must be made to tackle diverse failure scenarios. Fail-soft operation, a defining characteristic, refers to the system's ability to fail in a manner that safeguards as much functionality and data as possible. For instance, while a conventional UNIX system, upon detecting data corruption within the kernel, might display a failure message and terminate execution, a real-time system endeavors to rectify the issue or minimize its impact while continuing to operate. Typically, it notifies a user or process to initiate corrective measures and persists in operation, possibly at a reduced service level. Should a shutdown be necessary, efforts are made to sustain file and data consistency.

An essential facet of fail-soft operation is termed stability. A real-time system demonstrates stability if, in scenarios where meeting all task deadlines is impractical, it ensures the most critical, highest-priority task deadlines are met, even if some less crucial task deadlines occasionally fall behind.
Although a multitude of real-time OS designs cater to diverse real-time applications, several features are commonly shared among most real-time OSs:

- A more stringent utilization of priorities than standard OSs, coupled with preemptive scheduling tailored to meet real-time requisites.
- Bounded and relatively brief interrupt latency (the time between an interrupt's generation and its servicing).
- Enhanced timing predictability and precision compared to general-purpose OSs.

The crux of a real-time system is its short-term task scheduler. In its design, attributes like fairness and minimizing average response time take a back seat. What holds significance is the assurance that all hard real-time tasks meet (or commence) by their deadlines, and that a maximal number of soft real-time tasks also adhere to their deadlines.
Contemporary real-time operating systems often don't directly manage deadlines. Instead, they prioritize responsiveness to real-time tasks, ensuring that tasks can be swiftly scheduled as their deadlines approach. Real-time applications typically demand deterministic response times within the range of several milliseconds to sub-milliseconds, encompassing a wide array of scenarios; cutting-edge applications, such as military aircraft simulators, might even impose constraints in the 10-100 microsecond range.

3.4 Types of RTOS

Real-time operating systems (RTOS) can be categorized into several types based on their characteristics, scheduling algorithms, and intended use cases. Broadly they are classified as hard , soft and firm realtime OS as shown in figure 3.1 below.

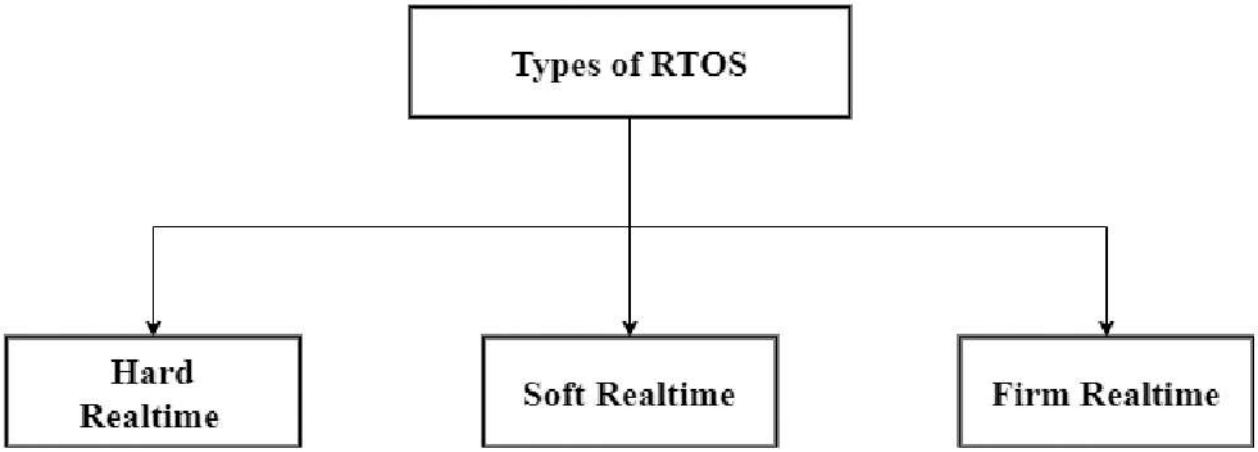

Figure 3.1:Types of RTOS

Hard Real-Time Operating System:
A hard real-time operating system is deployed in situations where tasks must be accomplished within predefined deadlines. If a task surpasses its allotted time, the entire system is considered to have failed. A prime example is a pacemaker, a medical device with insulated wires placed in the heart chambers to deliver electrical pulses and regulate heart rate. The precise timing in which these pulses are delivered necessitates a hard real-time operating system to ensure timely and critical operations.

Soft Real-Time Operating System:
In contrast, a soft real-time operating system is applied when slight variations in time are permissible. If a task slightly exceeds its specified time frame, it doesn't result in critical consequences. An illustration can be found in telephone switches, where the time taken to establish or terminate a call might vary slightly without leading to operational failure.

Firm Real-Time Operating System:
A firm real-time operating system occupies a middle ground between hard and soft real-time systems. It allows for occasional missed deadlines without causing complete failure, yet surpassing a certain threshold of missed deadlines could trigger a catastrophic system breakdown. Unlike hard real-time tasks that cause the system to fail if deadlines are breached, in a firm real-time environment, while tasks might exceed deadlines, the system remains operational; however, late results are discarded. A practical instance arises in video conferencing, where if an out-of-sequence frame arrives at the receiver, it's deemed redundant and discarded, demonstrating the discard aspect of a firm real-time system.

Each of these real-time operating system types caters to specific needs where the precision of task

execution and adherence to deadlines vary, offering flexibility and accommodation for different application requirements

3.5 RTOS In-Depth Overview

A real-time operating system (RTOS) brings a bunch of advantages to the table due to its advanced features and design approach. It uses a priority-based system to handle tasks effectively, making sure that critical tasks get priority over less urgent ones. This helps keep tasks organized and responsive. The RTOS also makes coding more efficient by providing ready-to-use functions through an Application Programming Interface (API). This means we can write code more easily and keep it neat and tidy. Moreover, it's designed in a way that reduces the complexity of managing timing requirements and promotes breaking down the work into separate tasks. This simplifies the development process and allows for testing individual parts independently. Operating system abstracts timing dependencies and adopts a task-centric design, reducing interdependencies between modules and facilitating modular development. On top of that, the RTOS works in a way that focuses computing power on the tasks that matter, which optimizes the system's performance and responsiveness. So, in simple words, RTOS helps us work smarter by managing tasks efficiently and making coding and collaboration smoother.

3.5.1 Components of RTOS

RTOS typically has the following components.

1. Scheduler:
   The scheduler plays a crucial role in ensuring that the highest priority process currently in execution is given precedence. It dictates the order and timing of task execution, ensuring that tasks with greater significance are addressed promptly. By managing priorities, the scheduler optimizes the utilization of system resources.

2. Critical Region:
   Critical regions are employed for seamless context switching, facilitated through mechanisms like Mutexes and Locks. These mechanisms guarantee that only one process at a time can access shared resources or sections of code, preventing conflicts and maintaining data integrity during concurrent execution.

3. Timer:
   Timers and clocks are employed to precisely monitor the execution time of processes. These tools enable the operating system to accurately measure the duration of task execution, aiding in meeting real-time requirements and scheduling tasks effectively.

4. Power Management:
   Efficient power management becomes pivotal in RTOS due to the presence of various devices with differing power needs. Effectively managing the power consumption of these devices ensures optimal energy utilization and extends the system's operational lifespan.

5. Communication Process:
   Communication processes are orchestrated through tools like Semaphores and Queues. These mechanisms determine the sequencing of processes, enabling orderly data exchange and synchronization among tasks.

6. Memory Management:
   Memory management within the context of a real-time operating system involves the allocation and utilization of virtual memory. Techniques such as paging are harnessed to optimize RAM usage by relocating processes not currently in use, resulting in effective resource allocation.

7. Peripheral Drivers:
   In real-time systems, external events are frequently linked to the system's operation. Peripheral drivers facilitate the flow of interrupts from external systems to the operating system, enabling seamless communication and interaction with external devices.

8. Device Management:
   The real-time operating system oversees a multitude of devices encompassing both internal and peripheral components. The coordination and synchronization of these devices are imperative for the system's proper functioning. Effective device management ensures harmony between various elements, allowing the system to execute its tasks with precision and efficiency.

3.5.2 Real-Time Operating System (RTOS) selection for a application.

Selecting a suitable Real-Time Operating System (RTOS) involves a careful evaluation of various factors:

- Security: Given that real-time applications often maintain internet connectivity, the paramount concern is security. The chosen RTOS must exhibit robust security features to safeguard against potential cyber threats and ensure data integrity.

- Compatibility with Environment: Seamless integration within the existing system environment is crucial. It's imperative to assess whether the selected RTOS aligns well with other interconnected systems. Opting for the latest RTOS versions with comprehensive support mechanisms can enhance interoperability.

- Feature Set: Before finalizing an RTOS, a thorough examination of its specifications is advisable. Factors such as memory allocation, resource prerequisites, task handling capabilities, and more should be carefully analyzed. The chosen RTOS should closely match the specific requirements of the application.

- Middleware Integration: The successful integration of an RTOS into the current system often requires middleware. Middleware acts as a bridge between the operating system and the

applications it supports. Choosing robust middleware can expedite integration processes, saving time and ensuring a smooth transition.

- Performance: As real-time applications hinge on instantaneous responsiveness, performance emerges as a pivotal criterion. The RTOS's performance capabilities significantly impact the overall system effectiveness. Optimal performance translates to a more efficient and reliable system operation.

In summation, the selection of an RTOS necessitates a comprehensive assessment of security provisions, compatibility, feature specifications, middleware support, and performance potential. Each factor plays a pivotal role in determining the suitability of the chosen RTOS for the intended application environment.

3.5.3 Example RTOS

Real-time operating systems find practical application in various domains, with the following examples illustrating their real-life usage:

- VxWorks: VxWorks is a renowned real-time operating system developed by Wind River Systems, widely used in automotive, consumer electronics, industrial devices, and networking equipment. The Wind microkernel unifies processes and threads as "tasks," simplifying task management. VxWorks features high performance, Unix-like multitasking, scalability, host and target-based development, and Device Software Optimization for improved software quality and reliability.

- RT Linux: RT Linux, denoting Real-time Linux, functions within the context of a Linux system. It serves as a bridge between the Linux environment and hardware, enhancing the real-time capabilities of Linux-based systems while maintaining compatibility with the underlying Linux framework.

- Lynx: Lynx, based on a microkernel architecture, is a Linux-compatible RTOS. Remarkably, programs developed for Linux can seamlessly operate on Lynx. This compatibility underscores the versatility of Lynx, enabling the execution of Linux binaries on its platform.

3.5.4 Advantages of Real-Time Operating Systems

Real-time operating systems offer several advantages that make them suited for specific applications:

- Ease of Use: Real-time applications are more accessible to develop and execute in RTOS environments. Complex problems can be compartmentalized into simpler tasks, rendering them easier to define, manage, and process.

- Efficient Resource Utilization: The kernel's adept resource management within RTOS results in

optimal utilization of resources like devices and systems. This efficiency enhances system performance.

- Compactness: RTOS systems are notably compact, consuming minimal space. This space efficiency is particularly advantageous, especially in scenarios with limited hardware resources.

- Embedded System Suitability: The compact nature of RTOS makes it well-suited for embedded systems, where space constraints are prevalent, and efficient real-time operations are essential.

- Error Management: RTOS systems typically offer a high degree of error management. The error manager comes into play whenever an error is detected, ensuring system stability and reliability.

3.5.5 Disadvantages of Real-Time Operating Systems

While RTOS possess several advantages, they also come with certain limitations:

- Cost: Hardware requirements like specialized device drivers can be costly, contributing to a higher overall system expense.

- Complexity: The algorithms and programs inherent to real-time systems tend to be intricate due to the need for precise timing and synchronization, increasing the complexity of development and maintenance.

- Limited Concurrent Tasks: RTOS systems may have limitations in terms of the number of tasks or programs that can be concurrently processed at any given time. This can impact the system's versatility in handling multiple tasks simultaneously.

3.6 Self-Assessment Questions

Q1. What distinguishes a real-time system from other computer systems, and why is meeting deadlines crucial in real-time computing? [4 marks, L2]

Q2. Define safety-critical systems in the context of real-time computing. Provide examples. [5 marks, L2]

Q3. Describe the evolution of real-time operating systems (RTOS) from specialized systems to more versatile and general-purpose solutions [5 marks, L2]

Q4. What are the different types of RTOS available, and why is having a variety of options beneficial for developers and users? [5 marks, L2]

Q5. How would you define a real-time system, and why is meeting deadlines crucial in this context? [5 marks, L2]

Q6. List three characteristics that distinguish real-time systems from normal desktop systems. [6 marks, L2]

Q7. Consider safety-critical systems such as flight management systems in airplanes. How might missed timing deadlines impact such systems?. [4 marks, L1]

Q8. Explain the difference between hard real-time and soft real-time systems using real-world examples. [4 marks, L1]

Q9. Explain how the concept of determinism in a real-time operating system ensures that tasks are completed within specific time intervals, even in a competitive resource-sharing environment. [6 marks, L2]

Q10. Describe how critical regions, facilitated through mechanisms like Mutexes and Locks, ensure that conflicts are prevented and data integrity is maintained during the execution of processes.[6 marks, L3]

Q11. Differentiate between user control in a real-time operating system and a non-real-time operating system.[5 marks, L3]

Q12. Define "fail-soft operation" and explain its significance in real-time systems.. [5 marks, L2]

3.7 Self-Assessment Activities

A1. Imagine you are a developer working on an autonomous car's collision avoidance system. The system must detect obstacles and apply brakes within milliseconds. Identify the type of real-time system required for the collision avoidance system: hard real-time or soft real-time. Explain your choice.

A2. You are tasked with designing an embedded system for a smart home security system. The system needs to monitor various sensors and trigger alarms if any suspicious activity is detected.Explain how the concept of an embedded system applies to this scenario. How does real-time computing enhance the security system's performance?

A3. You are designing an industrial robotic arm control system for a manufacturing plant. Precision and safety are paramount. Would you prioritize determinism or responsiveness for this real-time system? Justify your choice. How might user control play a role in the design of this robotic arm control system??

A4. You're working on an application where instantaneous responsiveness is crucial. Discuss the significance of performance in a real-time operating system and how it impacts the overall effectiveness and reliability of the system.

A5. You're responsible for designing an application where different tasks need to communicate and synchronize their execution. How can communication processes, involving tools like Semaphores and Queues, facilitate orderly data exchange and coordination among tasks in a real-time operating system?

3.8 Multiple-Choice Questions

Q1. What is a defining characteristic of a real-time system compared to other computer systems? [1 mark, L1]

A. Ability to multitask efficiently
B. High computational power
C. Precision in generating accurate results within a specific timeframe

D. Extensive memory capacity

Q2. An embedded system refers to: [1 mark, L1]

A. A computer system that focuses on graphical user interfaces
B. A component of a larger system with computing elements
C. A system that executes tasks in batches
D. A system with relaxed timing requirements

Q3. Which type of real-time system prioritizes critical tasks but is more flexible with timing deadlines? [1 mark, L1]

A. Hard real-time system
B. Soft real-time system
C. Batch-processing system
D. Safety-critical system

Q4. What essential functions do real-time operating systems (RTOS) offer? [1 mark, L1]

A. Gaming capabilities and multimedia support
B. Synchronization and graphics rendering
C. Scheduling, resource management, precise timing, and communication
D. Web browsing and social media integration

Q5. What is the benefit of having a diverse range of real-time operating systems (RTOS) in the market? [1 mark, L1]

A. It helps increase the price of RTOS options.
B. It allows developers to choose the same RTOS for different applications.
C. It enables users to select the cheapest option available.
D. It provides options to choose the best-suited RTOS for specific requirements.

Q6. Determinism in a real-time operating system refers to: [1 mark, L1]

A. The ability to execute tasks at arbitrary times.
B. The execution of tasks at fixed, predetermined intervals
C. The flexibility to handle unlimited process requests.
D. The ability to execute tasks without any priority.

Q7. Responsiveness in a real-time operating system primarily focuses on: [1 mark, L1]

A. Task execution time.
B. Task deadlines.
C. Interrupt handling delay after acknowledgment.
D. Task prioritization

Q8. A hard real-time operating system guarantees that: [1 mark, L1]

A. Critical tasks are completed within their deadlines.
B. Non-critical tasks are completed within their deadlines.
C. Tasks are completed based on user input.
D. Tasks are executed in a random order.

Q9. A firm real-time operating system: [1 mark, L1]

A. Requires all tasks to be completed within their deadlines..
B. Allows a few missed deadlines without total system failure.
C. Provides no flexibility in task scheduling.
D. Completely ignores task deadlines.

Q10. What is the role of the scheduler in a real-time operating system? [1 mark, L1]

A. It manages the power consumption of devices.
B. It handles communication processes using semaphores.
C. It determines the order and timing of task execution based on priorities.
D. It abstracts timing dependencies between modules.

Q11. How does an RTOS optimize system performance and responsiveness? [1 mark, L1]

A. By maximizing the use of external devices.
B. By prioritizing less critical tasks.
C. By distributing computing power evenly to all tasks.
D. By focusing computing power on the tasks that matter.

Q12. What does the term "fail-soft operation" mean in the context of a real-time system? [1 mark, L1]

A. The system immediately shuts down upon any failure.
B. The system handles failures by ignoring them.
C. The system minimizes the impact of failures and continues operation.
D. The system displays an error message and terminates execution.

3.9 Keys to Multiple-Choice Questions

Q1. Precision in generating accurate results within a specific timeframe (B)
Q2. A component of a larger system with computing elements (B)
Q3. Soft real-time system (B)
Q4. Scheduling, resource management, precise timing, and communication (C)
Q5. It provides options to choose the best-suited RTOS for specific requirements. (D)
Q6. The execution of tasks at fixed, predetermined intervals. (B)
Q7. Interrupt handling delay after acknowledgment. (C)
Q8. Critical tasks are completed within their deadlines. (A)
Q9. Allows a few missed deadlines without total system failure. (B)
Q10. It determines the order and timing of task execution based on priorities (C)
Q11. By focusing computing power on the tasks that matter. (D)
Q12. The system minimizes the impact of failures and continues operation. (C)

3.10 Summary of the Unit

The unit provides a comprehensive overview of the fundamental concepts, characteristics, components, and advantages of real-time operating systems. Real-time operating systems are specialized software systems designed to handle tasks within precise timeframes, making them indispensable in applications where timely and accurate execution is critical.The characteristics of real-time operating systems are explored in-depth, covering determinism, responsiveness, user control, reliability, and fail-soft operation. Determinism focuses on executing operations at predetermined intervals, and responsiveness deals with the system's delay in handling interrupts. User control in real-time systems is broader compared to conventional systems, enabling intricate task prioritization. Reliability is vital to avoid catastrophic

consequences, especially in safety-critical systems. Fail-soft operation ensures graceful degradation in the face of failures, maintaining functionality and data integrity.The unit delves into the components of real-time operating systems, elucidating the roles of the scheduler, critical regions, timer, power management, communication processes, memory management, peripheral drivers, and device management. These components collectively enable real-time systems to manage resources, handle tasks, and maintain synchronization and efficiency.

The selection of an appropriate RTOS for an application is discussed, considering factors such as security, compatibility, feature set, middleware integration, and performance. This selection process ensures that the chosen RTOS aligns with the application's requirements and environment.Advantages and disadvantages of real-time operating systems are highlighted. The benefits encompass ease of use, efficient resource utilization, compactness, suitability for embedded systems, and robust error management. On the flip side, challenges include higher costs, system complexity, and limitations in handling concurrent tasks.In conclusion, the unit provides a well-rounded understanding of real-time operating systems, their components, features, selection criteria, and real-world applications. This knowledge lays a strong foundation for further exploration and study of real-time systems in diverse contexts.

3.11 Keywords

Real-Time Operating System (RTOS), RTOS Characteristics, Determinism, Responsiveness, User Control, Reliability, Fail-Soft Operation

3.12 Recommended Learning Resources

[1] Silberschatz, A., Galvin, P., & Gagne, G. (2005). Operating System Concepts, 7th ed., Hoboken.

[2] William stalling. Operating Systems: Internal and design principles, 7th edition PHI

[3] D.M. Dhamdhere. Operating Systems: A concept-based Approach, 2nd Edition, TMH