RTOS Interview Questions and Answers For Freshers and Experienced

1) What is RTOS?

RTOS (Real-Time Operating System) is an operating system designed to run applications that require **detinistic timing** — meaning tasks must complete within specific deadlines.

Unlike general OS (Windows, Linux), RTOS guarantees response time.

Example:

- In a **car airbag system**, crash sensors must trigger airbag deployment within **~20–30 milliseconds**.
- An RTOS like **FreeRTOS** ensures the airbag task executes immediately without delay.

Types of real-time systems:

- **Hard Real-Time** – Missing deadline = system failure (Airbag, pacemaker)
- **Soft Real-Time** – Missing deadline reduces quality (Video streaming)
- **Firm Real-Time** – Occasional misses tolerated but not useful afterward

---

2) What is a Task in RTOS?

A **Task** is a small independent program (like a thread) that performs a specific function.

Each task has:

- Own stack
- Priority
- State (Ready, Running, Blocked)

Example (Smart Washing Machine):

- Task 1: Water Level Monitoring
- Task 2: Motor Control
- Task 3: Display Update

Each runs independently under RTOS control.

---

3) What is Task Priority?

Task Priority defines **which task gets CPU first**.

Higher priority task runs before lower priority task.

Example:
In a **medical ventilator**:

- Priority 3 – Oxygen Control (Highest)
- Priority 2 – Alarm System
- Priority 1 – LCD Display

If Oxygen Control becomes ready, it interrupts lower-priority tasks immediately.

---

4) What is Preemptive Scheduling?

In **Preemptive Scheduling**, the RTOS can interrupt a running task if a higher priority task becomes ready.

Used in most modern RTOS like **FreeRTOS**.

Example:

- Task A (Low priority) printing data.
- Emergency interrupt occurs → Task B (High priority).
- RTOS immediately stops Task A and runs Task B.

This ensures fast response time.

---

5) What is Context Switching?

Context Switching is the process of saving the current task's state and loading another task's state.

Saved information:

- Program Counter
- CPU Registers
- Stack Pointer

Example:
In a drone:

- Task 1: GPS Reading
- Task 2: Motor Speed Control

When switching between them, RTOS saves GPS task state and loads Motor Control task state.

Switch time: typically **1–10 microseconds** in microcontrollers.

---

6) What is a Tick Timer?

Tick Timer generates periodic interrupts (called system ticks).

Common tick rates:

- 1 ms (1000 Hz)
- 10 ms (100 Hz)

Used for:

- Task delays
- Time slicing
- Timeouts

Example:
If tick = 1 ms
`vTaskDelay(100)` → Task sleeps for 100 ms.

7) What is ISR in RTOS?

ISR (Interrupt Service Routine) is a function executed immediately when hardware interrupt occurs.

It should:

- Be short
- Not block
- Often signal a task

Example:
UART receives data → ISR triggered → ISR notifies Communication Task.

8) What is Stack in RTOS?

Each task has its own **stack memory**.

Stack stores:

- Local variables
- Function calls
- Return addresses

Example:
If task stack size = 1024 bytes
If exceeded → Stack overflow → System crash.

Typical embedded stack sizes:

- Small task: 512 bytes
- Complex task: 2048 bytes

9) What is Heap in RTOS?

Heap is shared dynamic memory used for:

- Creating tasks
- Queues
- Semaphores

Allocated using:
`pvPortMalloc()` (in **FreeRTOS**)

Example:
If total heap = 20 KB
Task1 = 2 KB
Queue = 1 KB
Remaining = 17 KB

---

10) What is Blocking State?

A task enters **Blocked State** when waiting for:

- Delay time
- Semaphore
- Queue data
- Event

Example:
Temperature Task waits for sensor data → blocked → CPU given to other tasks.

---

11) What is Ready State?

Task is ready to run but waiting for CPU.

If CPU free and highest priority → it runs.

Example:
Three tasks ready:

- P3 (High)
- P2 (Medium)
- P1 (Low)

RTOS runs P3 first.

---

12) What is Running State?

Task currently executing on CPU.

Only one task runs at a time (single-core MCU).

Example:
Motor Control Task updating PWM → Running State.

---

13) What is Cooperative Scheduling?

In Cooperative Scheduling:

- Tasks must voluntarily give up CPU.
- No forced preemption.

Used in older systems.

Example:
Task A must call `taskYIELD()` to allow Task B to run.

If Task A never yields → System freeze.

---

14) What is a Real-Time System?

A system where **correctness depends on both output and timing**.

Formula:
Correct Result + Correct Time = Correct System

Examples:

- Airbag system (deadline < 30 ms)
- Heart pacemaker
- Industrial robotic arm

---

15) What is RTOS Kernel?

RTOS Kernel is the core part that manages:

- Task scheduling
- Context switching
- Memory
- Interrupt handling

Examples of RTOS kernels:

- **FreeRTOS**
- **VxWorks**
- **ThreadX**

Kernel decides:

- Which task runs?
- When to switch?
- How to manage resources?

---

16) What is Semaphore?

A **Semaphore** is a signaling mechanism used to control access to shared resources or to synchronize tasks.

Types:

- **Binary Semaphore** (0 or 1)
- **Counting Semaphore** (0 to N)

It is mainly used for:

- Task synchronization
- Resource counting

Real-Time Example (Printer System):

Imagine 3 tasks want to use a printer.
A counting semaphore initialized to **1** ensures only one task prints at a time.

Semaphore = 1

Task A takes → 0 (others wait)

Task A releases → 1

In **FreeRTOS**, semaphores are commonly used between ISR and tasks.

Example:
UART interrupt gives semaphore → Communication task wakes up.

---

17) What is Mutex?

A **Mutex (Mutual Exclusion)** is a special locking mechanism used to protect shared resources.

Only **one task** can own a mutex at a time.

Important feature:
✔ Supports **Priority Inheritance**
✔ Has ownership (only owner can release)

Real-Time Example (I2C Bus Access):

Multiple tasks want to access I2C sensor.

- Task A locks mutex
- Task B waits
- Task A unlocks
- Task B gets access

This prevents data corruption.

---

18) What is the Difference Between Semaphore and Mutex?

Feature	Semaphore	Mutex
Purpose	Signaling & Resource counting	Protect shared resource
Ownership	No ownership	Has ownership
Priority Inheritance	No	Yes
Can ISR give?	Yes	No
Value Range	0 to N	Only 0 or 1

Example:

- Use **Semaphore** → Signal data ready
- Use **Mutex** → Protect shared memory

---

19) What is Priority Inversion?

Priority Inversion happens when:

- High-priority task waits for resource
- Low-priority task holds that resource
- Medium-priority task runs and delays low-priority task

So high-priority task indirectly waits for medium task.

Real Example (Mars Rover Incident):

NASA's **Mars Pathfinder** experienced priority inversion, causing system resets.

Scenario:

- Low task holds mutex
- High task needs it
- Medium task keeps running
- High task blocked → Problem

Solution: Priority Inheritance
Low-priority task temporarily gets high priority.

---

20) What is Deadlock?

Deadlock occurs when two or more tasks wait forever for each other's resources.

Condition Example:

Task A:

- Locks Resource 1
- Waits for Resource 2

Task B:

- Locks Resource 2
- Waits for Resource 1

Both wait forever → System stuck.

Real-world analogy:
Two cars entering narrow bridge from opposite sides.

Prevention:

- Resource ordering
- Timeout mechanism
- Avoid nested locks

---

21) What is Message Queue?

A **Message Queue** allows tasks to send data to each other safely.

Used for:

- Inter-task communication
- Data passing

Real-Time Example (Temperature Monitoring System):

- Sensor Task reads temperature
- Sends data to queue
- Display Task receives from queue

Queue size = 10 messages
If full → sender blocks (or fails)

In **FreeRTOS**, queues are widely used for safe communication.

---

22) What is Watchdog Timer?

A **Watchdog Timer (WDT)** is a hardware safety timer.

If system fails to reset it periodically → System automatically resets.

Real Example (Automotive ECU):

Engine Control Unit must reset watchdog every 100 ms.
If software crashes → No reset → WDT triggers system reboot.

Used in:

- Cars
- Medical devices
- Industrial machines

---

23) What is Time Slicing?

Time Slicing allows multiple tasks of **same priority** to share CPU.

Each task runs for fixed time slice (quantum).

Example:
Tick = 1 ms
Time slice = 5 ms

Task A runs 5 ms → Task B runs 5 ms → Task C runs 5 ms

Used in preemptive scheduling.

24) What is Multitasking?

Multitasking is the ability to run multiple tasks apparently at the same time.

In single-core MCU:

- Only one task runs at a time
- Fast switching creates illusion of parallelism

Example (Smart Home System):

- WiFi Task
- Sensor Task
- LCD Task
- Motor Control Task

RTOS switches between them quickly.

---

25) What is Hard Real-Time System?

A **Hard Real-Time System** is one where missing a deadline causes total system failure.

Deadline must ALWAYS be met.

Examples:

- Airbag system (< 30 ms response)
- Pacemaker
- Aircraft flight control

If response is late → catastrophic failure.

Used in:

- Aerospace
- Medical
- Defense
- Automotive safety systems

---

26) A High-Priority Task Is Missing Deadlines. How Will You Debug It?

Step-by-step Debug Approach:

1. **Check CPU Usage**
   - Is CPU overloaded?
   - Use runtime stats (`vTaskGetRunTimeStats()` in FreeRTOS)
2. **Check Blocking Calls**
   - Is task waiting on semaphore/queue?
   - Wrong timeout value?
3. **Check Priority Inversion**
   - Low task holding mutex?
   - Enable priority inheritance.
4. **Measure Execution Time**
   - Use GPIO toggle + oscilloscope.
   - Or cycle counter (DWT in Cortex-M).
5. **Check Interrupt Latency**
   - Long ISR blocking scheduler?

Real Example (Motor Control 1ms loop)

If control loop takes 1.5 ms → deadline miss.
Solution:

- Optimize math
- Move logging to lower priority
- Use fixed-point instead of floating-point

---

27) System Hangs After Running for Several Days. How Will You Debug It?

Likely causes:

- Memory leak
- Stack overflow
- Deadlock
- Heap fragmentation

Debug Plan:

1. Enable stack overflow hook
2. Monitor free heap periodically
3. Add watchdog reset logging
4. Use trace tools

In FreeRTOS:

- `configCHECK_FOR_STACK_OVERFLOW`
- `xPortGetFreeHeapSize()`

Real Case:

IoT device crashes after 5 days.
Cause → malloc used repeatedly without free.
Fix → Use static allocation.

---

28) Two Tasks Need to Share One Peripheral. How Will You Design It?

Best Design: **Single Driver Task + Queue**

Instead of:
Task A → UART
Task B → UART

Do:
Task A → Queue →
Task B → Queue →
UART Driver Task handles all access.

Alternative:
Use Mutex if short access.

Example:
I2C temperature + EEPROM both share bus.

---

29) RTOS System Consumes Too Much Power. How Will You Optimize It?

Power Optimization Steps:

1. Enable tickless idle
2. Put MCU in sleep mode
3. Reduce tick frequency (1000 Hz → 100 Hz)
4. Avoid busy waiting
5. Use event-driven design

In FreeRTOS:

- Enable `configUSE_TICKLESS_IDLE`

Real Example:

Battery IoT node:
Before → 20mA
After tickless → 200µA in sleep

Huge improvement.

---

30) How Will You Transfer Data from ISR to Task Safely?

Never:

- Call blocking APIs
- Use mutex inside ISR

Correct methods:

- Binary semaphore
- Queue (FromISR API)
- Direct-to-task notification

Example:

xQueueSendFromISR()

xSemaphoreGiveFromISR()

Use `portYIELD_FROM_ISR()` if needed.

---

31) Communication Task Is Blocking System Performance. How Will You Fix It?

Common problem:

- Long parsing
- Blocking send
- Large buffers

Fix Strategy:

1. Make it event-driven
2. Use DMA for UART/SPI
3. Reduce priority
4. Split into:
    - RX Task
    - Processing Task

5. Use circular buffers

Example:
Instead of blocking 50 ms send → use interrupt-based transmission.

---

32) How Will You Design a Real-Time Data Logging System?

Requirements:

- No data loss
- No timing impact

Design:
Sensor Task → Queue → Logger Task → SD card

Use:

- Double buffering
- DMA
- Low-priority logging

Critical rule:
Never write to SD in high-priority control task.

---

33) How Do You Debug Random Task Crashes in Production?

Possible reasons:

- Stack overflow
- Memory corruption
- Null pointer
- Race condition

Production Debug Methods:

1. Store crash logs in Flash
2. Enable stack watermark checking
3. Add assert hooks
4. Use watchdog reset reason register

Use tools like:

- SEGGER J-Link
- Percepio Tracealyzer

---

34) How Will You Manage Multicore RTOS Systems?

In SMP RTOS:

- Tasks can run on multiple cores.

Design considerations:

1. Core affinity
2. Shared memory protection
3. Inter-core communication (IPC)
4. Cache coherency

Example:
Core 0 → Control
Core 1 → UI + Logging

Use message buffers between cores.

---

35) Safety Task Is Delayed by Background Tasks. How Will You Fix It?

Steps:

1. Increase safety task priority
2. Make it highest priority
3. Reduce background priority
4. Remove blocking calls
5. Ensure ISR latency low

In Hard real-time:
Safety task must never wait.

---

36) How Will You Prevent Deadlocks in Large Systems?

Prevention rules:

1. Global resource ordering
2. Use timeout when taking mutex
3. Avoid nested locks
4. Keep critical sections short
5. Use deadlock detection tools

Design principle:
Never lock A→B in one place and B→A elsewhere.

37) How Do You Measure Real-Time Performance?

Tools & Methods:

1. GPIO toggle + Oscilloscope
2. CPU cycle counter
3. Trace tools
4. Measure:
    - Interrupt latency
    - Context switch time
    - Task execution time

Example:
Toggle pin at task start/end → measure exact execution (in µs).

---

38) How Will You Design a Fail-Safe RTOS System?

Techniques:

1. Watchdog timer
2. Redundant tasks
3. Safe state fallback
4. Health monitoring task
5. Brown-out detection
6. Periodic self-test

Example:
Industrial motor:
If control fails → Stop motor safely.

---

39) How Will You Optimize Memory in Low-RAM Systems?

Strategies:

1. Static allocation instead of dynamic
2. Reduce stack size using watermark
3. Use smaller data types
4. Avoid printf (huge memory)
5. Use memory pools
6. Remove unused libraries

Example:
MCU with 32KB RAM:
Careful task stack tuning saves 8–10 KB.

---

40) How Will You Implement OTA Updates in RTOS Devices?

Design:

1. Dual-bank firmware
2. Bootloader verification
3. CRC check
4. Rollback mechanism
5. Secure authentication

Flow:
Download → Verify → Switch partition → Reboot

Never:
Overwrite running firmware.

Example:
IoT device receives update via MQTT → stored in external flash → bootloader swaps image.

41) What is FreeRTOS?

FreeRTOS is a lightweight, open-source real-time operating system kernel designed for microcontrollers and small embedded systems.

It provides:

- Task scheduling
- Queues
- Semaphores & Mutexes
- Software timers
- Memory management

Used in:

- Automotive ECUs
- IoT devices
- Industrial controllers

Example: In an automotive ECU, FreeRTOS manages tasks like CAN communication, sensor reading, and motor control with deterministic timing.

---

42) What is a Task in FreeRTOS?

A **Task** is an independent thread of execution.

Each task has:

- Own stack
- Priority
- State (Ready, Running, Blocked)

Example (ECU):

- Task 1 → CAN RX
- Task 2 → Engine control loop (1 ms)
- Task 3 → Diagnostics

Created using:

xTaskCreate()

---

43) What is the FreeRTOS Scheduler? How Does It Work?

The scheduler decides which task runs.

Supports:

- Preemptive scheduling
- Time slicing
- Priority-based scheduling

How it works:

1. Highest priority Ready task runs.
2. If higher-priority task becomes ready → preemption occurs.
3. Tick interrupt drives scheduling decisions.

Example:
Engine Control Task (Priority 5) always preempts Logging Task (Priority 1).

---

44) What is a Queue in FreeRTOS?

A Queue is used for inter-task communication.

Features:

- Thread-safe
- Can send structures
- Works with ISR (`FromISR` APIs)

Example:
CAN ISR → sends frame to Queue → Processing Task reads it.

---

45) Difference Between `vTaskDelay()` and `vTaskDelayUntil()`

Function	Behavior	Use Case
`vTaskDelay()`	Delays relative to current time	General delays
`vTaskDelayUntil()`	Delays until absolute time	Periodic tasks

Example:
1 ms control loop → use `vTaskDelayUntil()` for precise periodic timing.

---

46) How to Design CAN Communication Task for Automotive ECU?

Design:

ISR:

- Receive CAN frame
- Push to Queue

CAN RX Task:

- Parse frame
- Validate ID
- Signal control task

CAN TX Task:

- Wait for message queue
- Transmit using driver

Use:

- High priority for safety signals
- Lower priority for diagnostics

Follow ISO 26262 safety layering.

---

47) Handling Safety-Critical Tasks Without Missing Deadlines

Steps:

1. Highest priority assignment
2. Use `vTaskDelayUntil()`
3. Avoid blocking APIs
4. Keep ISRs short
5. Measure WCET (Worst Case Execution Time)
6. Enable watchdog

Hard real-time rule:
Safety task must never wait for non-safety task.

---

48) How to Decide Task Priorities in Automotive Systems?

Method:

1. Identify deadlines
2. Identify task frequency
3. Use Rate Monotonic Scheduling principle

Example:

Task	Period	Priority
Engine Control	1 ms	Highest
Brake Monitor	5 ms	High
CAN	10 ms	Medium
Diagnostics	100 ms	Low

Shorter period → higher priority.

---

49) Memory Protection Between Safety and Non-Safety Tasks

Use:

- MPU (Memory Protection Unit)
- Separate memory regions
- Privileged/unprivileged tasks

On Cortex-M:
Use MPU wrappers in **FreeRTOS**

Safety task:

- Access to critical registers

Non-safety task:

- No access to safety memory

---

50) Watchdog Mechanism for ECU Fault Detection

Design:

1. Hardware watchdog enabled
2. Each safety task reports health
3. Supervisor task resets watchdog
4. If task hangs → watchdog not reset → system reboot

Add:

- Window watchdog
- Fault logging before reset

---

51) Designing RTOS Application for Multicore

Two models:

- AMP (Asymmetric Multi Processing)
- SMP (Symmetric Multi Processing)

Example:
Core 0 → Safety control
Core 1 → Infotainment

Use:

- Inter-core queues

- Shared memory with mutex

---

52) Reduce Power in RTOS-Based Mobile System

Techniques:

- Tickless idle
- Dynamic frequency scaling
- Sleep modes
- Disable unused peripherals
- Event-driven design

Example:
MCU sleeps between sensor readings → saves 90% battery.

---

53) Minimize Interrupt Latency in High-Performance SoCs

Steps:

1. Keep ISR short
2. Avoid nested interrupts
3. Reduce critical sections
4. Use fast interrupt (FIQ if available)
5. Use zero-latency interrupts

Measure latency with GPIO toggling.

---

54) Scheduling DSP and CPU Tasks

Common in TI processors.

CPU:

- Control + communication

DSP:

- Signal processing

Use:

- IPC mechanism
- Shared buffers
- Mailbox interrupts

55) Handle Shared Memory Across Cores Safely

Use:

- Cache coherency management
- Memory barriers
- Mutex or spinlocks
- Atomic operations

Critical for avoiding race conditions.

---

56) Integrate Embedded Linux with RTOS

Common in industrial gateways.

Example:

- Linux → UI + Networking
- RTOS → Real-time control

Use:

- Shared memory
- RPMsg
- Ethernet bridge

---

57) Hybrid Scheduling Between Linux and RTOS

Linux handles:

- Non-deterministic tasks

RTOS handles:

- Hard real-time tasks

Communication via:

- IPC
- Shared DDR memory

Linux cannot guarantee microsecond deadlines; RTOS can.

---

58) Optimize Firmware for Deterministic Performance

Steps:

- Remove dynamic memory
- Avoid printf
- Use fixed-point math
- Reduce interrupt nesting
- Measure WCET

Determinism > raw speed.

---

59) Design Real-Time Device Drivers in RTOS

Rules:

1. ISR handles minimal work
2. Driver task handles heavy work
3. Use DMA if possible
4. Protect hardware with mutex

Never block inside ISR.

---

60) Manage Boot Sequence in RTOS + Linux Platform

Typical flow:

Bootloader →
RTOS core init →
Linux kernel load →
IPC setup →
Application start

Ensure:

- Safety RTOS starts before Linux

---

61) How is TI-RTOS Different?

TI-RTOS:

- Integrated tightly with TI hardware
- Includes drivers + middleware
- Supports BIOS kernel (SYS/BIOS)

Compared to generic RTOS:

- More hardware-optimized
- Vendor ecosystem integrated

---

62) Schedule DSP Tasks with Strict Timing

Use:

- Hardware timers
- Dedicated DSP core
- Fixed scheduling slots
- Avoid OS jitter

Use double buffering for continuous data.

---

63) Manage Peripherals Safely in RTOS

Options:

- Single driver task model (recommended)
- Mutex protection
- Avoid direct peripheral access from multiple tasks

Prevents race conditions.

---

64) Tune ISRs for Real-Time Constraints

Best practices:

- Keep < 5–10 µs
- No blocking
- Defer processing to task
- Avoid heavy loops

Measure latency regularly.

---

65) Low Power in TI RTOS Applications

Use:

- Idle hooks
- Power manager APIs
- Peripheral gating
- Deep sleep modes

Common in battery-operated TI SoCs.

---

66) Design RTOS on STM32 Using FreeRTOS

On **STM32**:

1. Configure in STM32CubeMX
2. Define tasks
3. Set priorities
4. Configure heap
5. Enable MPU if needed
6. Enable tickless mode

Example:
Motor control → High priority
UART logging → Low priority

---

67) Integrate DMA with RTOS Without Corruption

Steps:

1. Use double buffers
2. Protect buffer pointers
3. Use cache maintenance (if Cortex-M7)
4. Signal task using semaphore

DMA ISR → Give semaphore → Processing task runs.

---

68) Handle HAL Drivers in Multithreaded RTOS

HAL not always thread-safe.

Solutions:

- Wrap HAL calls with mutex
- Single driver task model
- Avoid blocking HAL inside ISR

---

69) Low Power in STM32 RTOS Systems

Use:

- Stop mode
- Sleep mode

- Tickless idle
- Wakeup via interrupt
- Disable unused clocks

On **STM32**, STOP mode reduces current to µA range.

---

70) Debug Deadlocks and Priority Issues in FreeRTOS

Steps:

1. Enable stack overflow detection
2. Use trace tools (like Tracealyzer)
3. Monitor mutex ownership
4. Add timeout for locks
5. Check task states

Common causes:

- Nested locks
- Missing `xSemaphoreGive`
- Wrong priority design

71) What is Event Group in RTOS?

An **Event Group** allows a task to wait for multiple events using bit flags.

Each bit represents an event.

Used for:

- Multi-condition synchronization
- State signaling

Example (Industrial Machine):

- Bit 0 → Motor Ready
- Bit 1 → Temperature OK
- Bit 2 → Safety Door Closed

Control task waits until all bits set before starting machine.

---

72) What is Direct-to-Task Notification?

A lightweight RTOS mechanism for fast task signaling.

Advantages:

- Faster than semaphore
- Uses less RAM
- Ideal for ISR-to-task communication

Example:
ADC ISR completes conversion → notifies Processing Task directly.

Used heavily in performance-critical systems.

---

73) What is Zero-Latency Interrupt?

A special interrupt that bypasses RTOS kernel masking.

Used for:

- Ultra-critical timing
- Safety shutdown signals

Example:
Overcurrent detection interrupt immediately disables PWM without waiting for scheduler.

---

74) What is Worst Case Execution Time (WCET)?

WCET is the maximum time a task can take to execute.

Critical in hard real-time systems.

How to measure:

- Cycle counter
- Oscilloscope with GPIO toggle
- Static timing analysis tools

Example:
Motor control loop must finish in 1 ms.
Measured WCET = 800 µs → Safe.

---

75) What is Jitter in RTOS?

Jitter is variation in task execution timing.

Causes:

- Interrupt delays
- Cache effects
- Shared resource blocking

Example:
Control loop expected every 1 ms.
Actual execution: 1 ms ± 50 µs → 50 µs jitter.

Too much jitter affects control stability.

---

76) What is Rate Monotonic Scheduling (RMS)?

A fixed-priority scheduling algorithm.

Rule:
Shorter period → Higher priority.

Used when:
Tasks are periodic and independent.

Example:
1 ms task → Highest priority
10 ms task → Lower priority

Common in automotive ECUs.

---

77) What is Earliest Deadline First (EDF)?

A dynamic scheduling algorithm.

Rule:
Task with nearest deadline runs first.

More CPU efficient than RMS.

Used in:
Advanced RTOS or academic systems.

---

78) What is Memory Fragmentation in RTOS?

Occurs when heap memory becomes divided into small unusable blocks.

Caused by:
Frequent malloc/free.

Problems:
Allocation failure even when total memory available.

Solution:

- Static allocation
- Memory pools
- Fixed block allocator

---

79) What is a Software Timer in RTOS?

A timer managed by RTOS kernel.

Used for:

- Periodic callbacks
- Timeout handling
- Delayed execution

Example:
Heartbeat LED toggle every 500 ms.

Runs in timer service task context.

---

80) What is Critical Section?

A code region where interrupts are temporarily disabled.

Used to protect shared resources.

Example:
Updating shared counter variable.

Must be:
Short duration
Minimal blocking

Long critical sections increase interrupt latency.

---

81) What is Tickless Mode?

In tickless mode, periodic tick interrupt stops during idle.

Benefits:

- Reduced power consumption
- Less CPU wake-ups

Example:
Battery IoT node sleeps for 5 seconds without 1 ms tick wakeups.

82) What is Run-Time Statistics in RTOS?

Used to measure CPU usage of each task.

In FreeRTOS:
vTaskGetRunTimeStats()

Helps identify:

- CPU hogging task
- Starvation problems

83) What is Stack Overflow Detection?

RTOS can detect when a task exceeds its stack limit.

Methods:

- Canary pattern check
- MPU protection

In FreeRTOS:
configCHECK_FOR_STACK_OVERFLOW

Prevents silent memory corruption.

84) What is Starvation in RTOS?

Occurs when low-priority task never gets CPU time.

Cause:
High-priority tasks running continuously.

Solution:

- Time slicing
- Proper priority design
- Yielding when possible

85) What is CPU Load in RTOS?

Percentage of CPU time used by active tasks.

High CPU load (>90%) may cause:

- Missed deadlines
- Increased latency

Measured using:
Idle task runtime monitoring.

---

86) What is Safe State in Embedded Systems?

A predefined system condition when fault occurs.

Example:
Industrial motor fault → Stop motor + activate brake.

RTOS must quickly switch to safe state upon error.

---

87) What is Brown-Out Detection?

Detects low supply voltage condition.

Prevents:
Flash corruption
Unexpected behavior

Common in automotive ECUs.

---

88) What is Deterministic Behavior?

System produces predictable timing behavior.

Key requirement in:

- Automotive
- Medical
- Aerospace

Achieved by:

- Fixed priorities
- No dynamic allocation
- Controlled interrupts

---

89) What is Control Loop in RTOS?

A periodic task performing feedback control.

Example:
PID motor control every 1 ms.

Must:
Use vTaskDelayUntil()
Avoid blocking calls
Maintain low jitter

---

90) What is Latency vs Throughput?

Latency:
Time to respond to event.

Throughput:
Amount of work done per unit time.

Real-time systems prioritize:
Low latency over high throughput.

---

91) What is Inter-Process Communication (IPC)?

Mechanism for tasks/cores to exchange data.

Types:

- Queue
- Semaphore
- Shared memory
- Mailbox

Used heavily in multicore systems.

---

92) What is Cache Coherency Issue?

In multicore systems:
Each core may cache shared memory.

Without coherency:
One core sees old data.

Solution:

- Cache flush/invalidate
- Memory barriers

Critical in ARM Cortex-A systems.

---

93) What is Bootloader in RTOS Systems?

Small program that runs before main firmware.

Functions:

- Firmware update
- Image verification
- Secure boot

Used in OTA systems.

---

94) What is Secure Boot?

Ensures only authenticated firmware runs.

Uses:

- Digital signature
- Cryptographic verification

Important in:
Automotive
Medical
IoT security

---

95) What is Fault Tolerance?

System continues operating even when faults occur.

Methods:

- Redundant tasks
- Watchdog
- Error recovery logic

Used in aerospace systems.

---

96) What is Task Affinity in Multicore RTOS?

Defines which core a task runs on.

Example:
Core 0 → Safety tasks only
Core 1 → UI tasks

Improves predictability.

---

97) What is ISR Latency?

Time between interrupt event and ISR execution.

Affected by:

- Interrupt masking
- Critical sections
- CPU load

Measured in microseconds.

---

98) What is DMA in RTOS Systems?

Direct Memory Access transfers data without CPU involvement.

Used for:

- UART
- SPI
- ADC
- Ethernet

Improves performance and reduces CPU load.

---

99) What is Health Monitoring Task?

A supervisory task that checks:

- Task alive status
- CPU load
- Stack usage

Feeds watchdog only if system healthy.

Common in safety systems.

---

100) What is Real-Time System Validation?

Process of verifying system meets timing requirements.

Includes:

- Stress testing
- WCET validation
- Interrupt latency measurement
- Long-duration reliability testing

Final goal:
Ensure deterministic behavior under all conditions.
