

# RealTime Operating Systems Proficiency Test

Embedded  
Expert IO

# TABLE OF CONTENT

---

About Us .....	01
Learning Outcomes .....	02
Questions	
Introduction to Real-time Operating Systems .....	07
The Internals of Realtime Operating Systems .....	08
Threads and Thread Control Blocks(TCB) .....	13
The Scheduler and Scheduling Algorithms .....	15
Inter-Thread Communication .....	19
Developing Realtime Operating Systems .....	21
FreeRTOS .....	33
CMSIS-RTOS APIs .....	42
<b>Answer Sheet .....</b>	<b>44</b>

# About Us

EmbeddedExpertIO stands as a premier source of tailored embedded systems development courses, catering to individuals and enterprises seeking to hone or acquire embedded firmware programming expertise. Our extensive course selections encompass beginner to advanced levels, addressing diverse facets of embedded systems development, such as WiFi, STM32 Bare-Metal, WiFi, Ethernet, GSM and beyond.

Our core objective is to equip individuals and organizations with the indispensable skills to thrive in the swiftly evolving embedded systems sector. We achieve this by providing immersive, hands-on education under the guidance of seasoned industry specialists. Our ambition is to emerge as the favored learning platform for embedded systems development professionals across the globe.

34A Frithville Gardens,  
London, W12 7JN  
England, United Kingdom

**e:**[support@embeddedexpert.io](mailto:support@embeddedexpert.io)  
**https://**[embeddedexpert.io](https://embeddedexpert.io)

# Learning Outcomes

At EmbeddedExpertIO, our aim is to prepare our students comprehensively for the challenges they'll face in the job market. To ensure your success in securing Embedded Software Engineer roles at leading companies we've identified several key knowledge areas and skills that are critical to demonstrate during the interview process.

## **Understanding of Embedded Systems**

Candidates should have a strong foundational understanding of embedded systems. This includes knowledge about how these systems are designed, how they interact with hardware and software, and how they operate within constraints such as limited power or memory. They should be well-versed in concepts such as real-time operating systems, microcontroller architectures, memory management, and system optimization.

# Proficiency in Relevant Programming Languages

Proficiency in languages commonly used in embedded systems, particularly C and C++, is essential. Candidates should be able to write efficient, readable, and maintainable code. Understanding of Assembly language could also be advantageous for certain roles.

# Familiarity with Relevant Tools and Protocols

This is a critical skill for any engineering role. Candidates should be adept at identifying problems, analyzing potential solutions, and implementing these solutions effectively. They should be able to demonstrate this ability in both theoretical and practical terms.



# Knowledge of Testing and Debugging

Understanding how to test and debug embedded software is crucial. Candidates should be comfortable with techniques for unit testing, integration testing, and system testing. They should also know how to use debugging tools effectively and understand common debugging techniques.

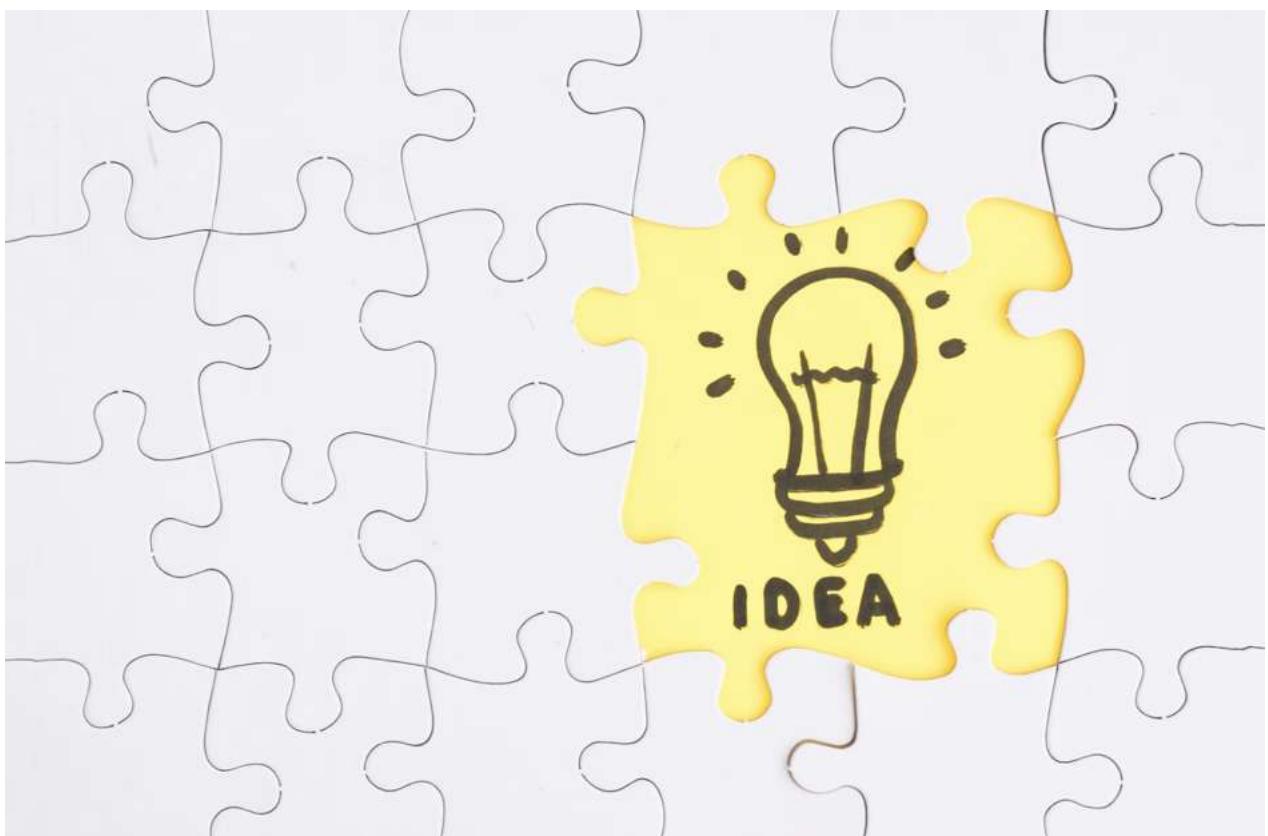
# Programming Proficiency

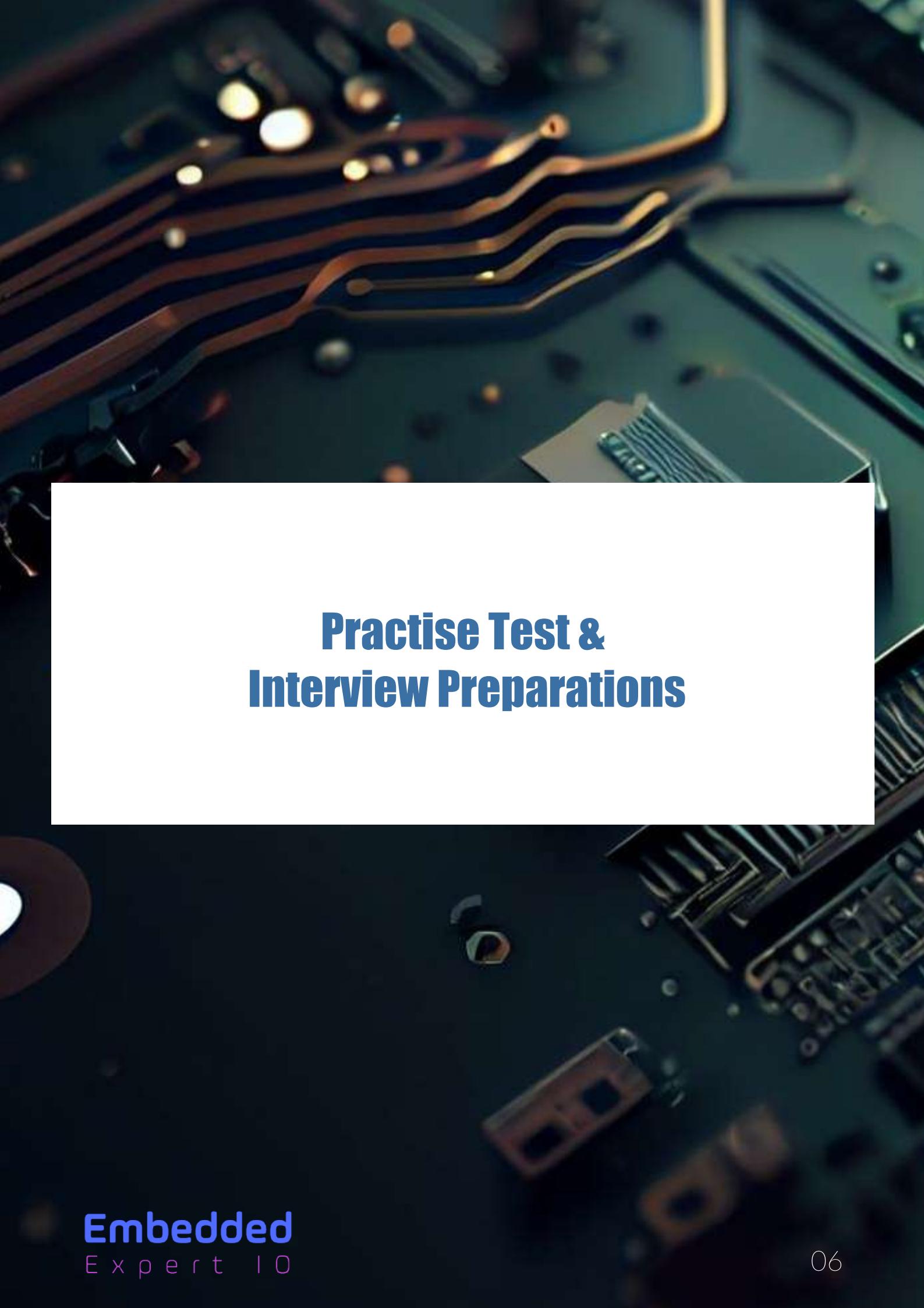
Candidates must be able to understand and develop low level code for common architectures such as ARM.

By mastering these areas, students will be well-prepared to showcase their technical prowess, their problem-solving abilities, and their readiness to contribute effectively in a Embedded Software Engineer role. Our curriculum at EmbeddedExpertIO is meticulously designed to help students develop these competencies, giving them the confidence to excel in their interviews and their subsequent professional roles.

# Problem-Solving Skills

This is a critical skill for any engineering role. Candidates should be adept at identifying problems, analyzing potential solutions, and implementing these solutions effectively. They should be able to demonstrate this ability in both theoretical and practical terms.





# Practise Test & Interview Preparations

# Introduction to Real-time Operating Systems

## Question 1

What is the key difference between a real-time operating system (RTOS) and a general-purpose operating system (GPOS)?

- A) A RTOS has more complex tasks
- B) A RTOS guarantees that it will meet a deadline
- C) A RTOS uses memory
- D) A RTOS requires multiple processors

## Question 2

What are the two most common performance metrics for real-time operating systems?

- A) Deadlines and reliability
- B) Speed and capacity
- C) Efficiency and effectiveness
- D) Complexity and simplicity

## Question 3

Question: Which performance metric measures whether an RTOS is running in a predictable way with a guaranteed response?

- A. Deadlines
- B. Reliability
- C. Efficiency
- D. Scalability

# The Internals of Realtime Operating Systems

## Question 4

What are the two main stack operations?

- A. Pop and Peek.
- B. Push and Pop.
- C. Peek and Purge.
- D. Purge and Push.

## Question 5

When data is read from the stack, which element is read first?

- A. The first element to be pushed onto the stack.
- B. The last element to be pushed onto the stack.
- C. The element in the middle of the stack.
- D. The element at the bottom of the stack.

## Question 6

What does the term "context saving" mean in relation to ARM Cortex architecture and exceptions?

- A. Saving all the active files on your desktop before an exception occurs.
- B. Saving the current state of the system to a physical storage device for later recovery.

- C. Storing the contents of specific registers into the stack when an exception occurs, allowing the program to continue from where it left off before the exception.
- D. Sending a snapshot of the current system state to a remote server for data analysis.

### **Question 7**

How is the stack pointer register used in the context of exceptions in ARM Cortex architecture?

- A. It points to the bottom of the stack in RAM.
- B. It points to the top of the stack in RAM, used to locate the top of the stack.
- C. It saves the current state of all the registers when an exception occurs.
- D. It points to the exception handler function in the system memory.

### **Question 8**

How does manipulating the contents of the Program Counter (PC) register during an exception affect the flow of the program?

- A. It has no effect on the flow of the program.

- B. It can cause the program to jump to a new location after the exception is handled rather than continuing where it left off.
- C. It pauses the execution of the program until the exception is handled.
- D. It resets the PC to the beginning of the program, causing the program to start over.

### **Question 9**

What is the role of the stack pointer?

- A. It always points to the bottom of the stack.
- B. It always points to the middle of the stack.
- C. It always points to the top of the stack.
- D. It moves randomly within the stack.

### **Question 10**

In a Cortex M processor, why does the stack pointer decrement by four after a push operation?

- A. Because it operates on 32-bit data and 4 bytes equals 32 bits.
- B. Because it operates on 16-bit data.
- C. Because it operates on 64-bit data.
- D. Because it operates on 8-bit data.

### **Question 11**

In the context of a Cortex M processor, what is the SP?

- A. Stack Popper.
- B. Stack Pointer.
- C. Stack Processor.
- D. Stack Pusher.

### **Question 12**

What does the term 'Last-In-First-Out' mean in the context of stack data storage?

- A. The first element to be pushed onto the stack is the first to be read.
- B. The last element to be pushed onto the stack is the first to be read.
- C. All elements are read simultaneously.
- D. The middle element to be pushed onto the stack is the first to be read.

### **Question 13**

Which register is known as the stack pointer in a Cortex M processor?

- A. Register 4
- B. Register 5
- C. Register 13
- D. Register 6

### **Question 14**

How many stack pointers does a Cortex-M microcontroller have?

- A. One
- B. Two
- C. Three
- D. Four

### **Question 15**

What does CPU utilization measure?

- A. The percentage of available CPU cycles actually being used
- B. The number of tasks completed per unit time
- C. The time it takes for each task to complete
- D. The percentage of time the CPU spends in the idle state

### **Question 16**

What does "blocking code" refer to?

- a. Code that prevents the program from running.
- b. Code that spends a lot of time in delay function, effectively blocking the execution of other operations.
- c. Code that contains errors and causes the program to crash.
- d. Code that restricts certain functions from being executed.

### **Question 17**

In an autonomous vehicle, why could a busy wait solution be problematic?

- a) It could lead to outdated information and potentially accidents
- b) It would make the vehicle too slow
- c) It would use too much battery power
- d) It could overload the sensors

Threads and Thread Control Blocks (TCB)

### **Question 18**

What does TCB stand for in the context of thread management?

- A. Thread Control Block
- B. Time Cycle Buffer
- C. Thread Context Backup
- D. Task Control Buffer

### **Question 19**

What are the two compulsory parameters in a thread control block?

- A. Stack pointer and thread ID
- B. Thread status and priority
- C. Stack pointer and next thread pointer
- D. Thread period and burst time

### **Question 20**

What type of data structure is typically used to link thread control blocks?

- A. Array
- B. Linked list
- C. Queue
- D. Stack

### **Question 21**

What is the purpose of the next pointer (nextPt) parameter in a thread control block?

- A. To store the thread status
- B. To hold the thread ID
- C. To link to the next thread control block
- D. To indicate the thread's burst time

### **Question 22**

Which type of threads execute frequently but their runtime cannot be anticipated?

- A. Sporadic threads
- B. Periodic threads
- C. Aperiodic threads
- D. Main threads

### **Question 23**

What is the role of the thread control block (TCB) in thread management?

- A. To store the thread's program code
- B. To allocate memory for the thread's stack
- C. To hold the thread's execution context and information
- D. To determine the thread's priority in the scheduling algorithm

The Scheduler and Scheduling Algorithms

### **Question 24**

What is the main difference between processes and threads?

- A. Processes are part of a thread, but threads are not part of a process
- B. Threads are part of a process, but processes are not part of a thread
- C. Processes execute tasks, while threads execute programs
- D. Threads are lightweight processes, while processes are heavyweight threads

### **Question 25**

What is the purpose of the thread scheduler in a real-time system?

- A) To allocate resources to different tasks
- B) To execute each task as a main task in a somewhat parallel mode
- C) To check the status of each task
- D) To organize the sequence of tasks

### **Question 26**

How are scheduling algorithms classified?

- A. Static vs. dynamic and preemptive vs. non-preemptive
- B. Time-triggered vs. event-triggered and periodic vs. aperiodic
- C. Shortest job first vs. round-robin and rate monotonic vs. weighted round-robin
- D. First come first served vs. round-robin and shortest job first vs. rate monotonic

### **Question 27**

What is the characteristic of the round-robin (RR) scheduler?

- A. It is a non-preemptive scheduler with priority-based execution
- B. It executes tasks based on their burst time
- C. It allows threads to complete execution before switching to the next thread
- D. It uses a time quantum for each thread's execution

### **Question 28**

What is the purpose of the thread scheduler?

- A. To handle the removal of running threads from the CPU
- B. To select another thread based on a particular algorithm
- C. To run a number of threads ready to be executed
- D. All of the above

### **Question 29**

What is the purpose of the time quanta in round-robin scheduling?

- A. To determine the order of thread execution
- B. To assign weights to threads
- C. To get the time of the system
- D. To limit the execution time of each thread

### **Question 30**

Which scheduling algorithm does the round-robin scheduler resemble when the quanta size is extremely large?

- A. First-Come First-Served (FCFS) scheduler
- B. Shortest Job Next (SJN) scheduler
- C. Priority-based scheduler
- D. Weighted Round Robin (WRR) scheduler

### **Question 31**

What is the recommended rule of thumb when choosing a time quanta for round-robin scheduling?

- A. Choose a time quanta equal to the context switch time
- B. Choose a time quanta significantly larger than the context switch time
- C. Choose a time quanta slightly smaller than the context switch time
- D. Choose a time quanta equal to the number of threads

### **Question 32**

In the weighted round-robin scheduler, how can the importance of a thread be represented?

- A. By assigning a weight parameter to each thread
- B. By varying the burst time of each thread
- C. By adjusting the time quanta for each thread
- D. By arranging the threads in a circular linked list

### **Question 33**

What is the purpose of the context switching process in the round-robin scheduler?

- A. To assign weights to threads based on their importance
- B. To determine the order of thread execution
- C. To save and restore the execution state of threads
- D. To allocate a time quanta for each thread

### **Question 34**

How does the round-robin scheduler ensure fairness among threads?

- A. It assigns a higher priority to longer-running threads
- B. It assigns a fixed time quanta to each thread
- C. It dynamically adjusts the time quanta based on thread performance
- D. It prioritizes threads based on their initialization order

## Inter-Thread Communication

### **Question 35**

Which mechanism is used for safe communication and synchronization between tasks/interrupts without carrying additional data?

- A) Semaphore
- B) Mutex
- C) Counting Semaphore
- D) Queue

### **Question 36**

What is the typical design pattern for using a binary semaphore in an RTOS?

- A) "Take" the semaphore before the critical section and "give" it right after

- B) "Take" the semaphore in the same task as the critical section
- C) Increment the semaphore counter before using the shared resource
- D) Use a switch statement to handle different message codes in the receiver task

### **Question 37**

What is the purpose of a counting semaphore in an RTOS?

- A) To trigger activation of a task
- B) To protect a critical section between tasks
- C) To keep track of limited shared resources
- D) To pass arbitrary messages to tasks

### **Question 38**

What is the purpose of a semaphore?

- A) To manage shared resources
- B) To synchronize tasks
- C) To signal another task
- D) All of the above

## Developing Realtime Operating Systems

```
1 #include "timebase.h"
2 #include "stm32f4xx.h"
3
4 #define ONE_SEC_LOAD 16000000
5
6 #define CTRL_ENABLE (1U<<0)
7 #define CTRL_TICKINT (1U<<1)
8 #define CTRL_CLKSRC (1U<<2)
9 #define CTRL_COUNTFLAG (1U<<16)
10
11 #define MAX_DELAY 0xFFFFFFFFU
12
13 volatile uint32_t g_curr_tick;
14 volatile uint32_t g_curr_tick_p;
15
16 volatile uint32_t tick_freq = 1;
17
18 /*Delay in seconds*/
19 void delay(uint32_t delay)
20 {
21     uint32_t tickstart = get_tick();
22     uint32_t wait = delay;
23
24     if(wait < MAX_DELAY)
25     {
26         wait += (uint32_t)(tick_freq);
27     }
28
29     while((get_tick() - tickstart) < wait){}
30 }
31
32 void tick_increment(void)
33 {
34     g_curr_tick += tick_freq;
35 }
```

```

36
37     uint32_t get_tick(void)
38 {
39     __disable_irq();
40     g_curr_tick_p = g_curr_tick;
41     __enable_irq();
42
43     return g_curr_tick_p;
44 }
45
46 void timebase_init(void)
47 {
48     /*Reload the timer with number of cycles per second*/
49     SysTick->LOAD = ONE_SEC_LOAD - 1;
50
51     /*Clear Systick current value register*/
52     SysTick->VAL = 0;
53
54     /*Select internal clock source*/
55     SysTick->CTRL = CTRL_CLKSRC;
56
57     /*Enable interrupt*/
58     SysTick->CTRL |= CTRL_TICKINT;
59
60     /*Enable systick*/
61     SysTick->CTRL |= CTRL_ENABLE;
62
63     /*Enable global interrupts*/
64     __enable_irq();
65 }
66
67 void SysTick_Handler(void)
68 {
69     tick_increment();
70 }

```

### **Question 39**

- What is the purpose of the time base initialization function?
- A. To initialize the systick timer to operate in interrupt mode and create a timeout each second.
  - B. To load values into the systick reload value register.
  - C. To clear the systick current value register.
  - D. All of the above.

### **Question 40**

Which timer is used to create a time base because of its consistency across all Arm Cortex microcontrollers?

- A. Timer0
- B. Systick Timer
- C. Timer1
- D. Real-Time Clock Timer

### **Question 41**

If you want to create a timeout to occur in a millisecond, how many cycles should you load into the systick reload value register given a default system frequency of 16Mhz?

- A. 16 million
- B. 16
- C. 16,000
- D. 1 million

### **Question 42**

What is the function of the "get\_tick" function?

- A. It disables global interrupts and gets the tick count.
- B. It enables global interrupts and gets the tick count.
- C. It disables global interrupts, gets the tick count, and then enables global interrupts.
- D. It simply returns the tick count.

### **Question 43**

How often is the Tick\_increment function called?

- A. Every microsecond
- B. Every millisecond
- C. Every second
- D. Whenever the get\_tick function is called

### **Question 44**

What is the purpose of the delay function in the given code?

- A. It waits for a specific number of ticks to occur.
- B. It increments the tick count.
- C. It initializes the systick timer.
- D. It retrieves the current tick count.

### **Question 45**

What is the purpose of the SysTick\_Handler function in this code?

- A. It is automatically called when a systick interrupt occurs and increments the tick count.
- B. It is used to manually trigger a systick interrupt.
- C. It initializes the systick timer.
- D. It is called to get the current tick count.

### **Question 46**

Why is the tick\_freq variable declared as volatile?

- A. To ensure the compiler does not optimize it out.
- B. To allow it to be modified by multiple threads.
- C. To signal that it may be externally modified outside normal program flow.
- D. All of the above.

### **Question 47**

Why is the systick priority set to a low value in a RTOS?

- A. To enable important hardware interrupts to execute
- B. To ensure the scheduler has the highest priority
- C. To synchronize the systick interrupt with the system clock
- D. To prevent other interrupts from occurring

## Question 48

What is the purpose of the \_\_asm("CPSIE I") instruction ?

- A. Loading the address of the current Pt into R1
- B. Disabling global interrupts
- C. Enabling global interrupts
- D. Saving the context of the currently running thread

## Question 49

What does the T bit in the PSR register represent?

- A. Thread status flag
- B. Time quanta value
- C. Task priority level
- D. Thumb instruction set flag

```
1 #include "osKernel.h"
2
3 #define NUM_OF_THREADS      3
4 #define STACKSIZE          400
5
6 #define BUS_FREQ            16000000
7
8 #define CTRL_ENABLE         (1U<<0)
9 #define CTRL_TICKINT        (1U<<1)
10 #define CTRL_CLKSRC         (1U<<2)
11 #define CTRL_COUNTFLAG     (1U<<16)
12 #define SYSTICK_RST         0
13
```

```

13
14     void osSchedulerLaunch(void);
15
16     uint32_t MILLIS_PRESCALER;
17     ∵ struct tcb {
18         int32_t *stackPt;
19         struct tcb *nextPt;
20     };
21
22     typedef struct tcb tcbType;
23
24     tcbType tcbs[NUM_OF_THREADS];
25     tcbType *currentPt;
26
27     int32_t TCB_STACK[NUM_OF_THREADS][STACKSIZE];
28
29     ∵ void osKernelStackInit(int i) {
30         tcbs[i].stackPt = &TCB_STACK[i][STACKSIZE - 16];
31         TCB_STACK[i][STACKSIZE - 1] = (1U<<24);
32         TCB_STACK[i][STACKSIZE-3] = 0xAAAAAAA;
33         TCB_STACK[i][STACKSIZE-4] = 0xAAAAAAA;
34         TCB_STACK[i][STACKSIZE-5] = 0xAAAAAAA;
35         TCB_STACK[i][STACKSIZE-6] = 0xAAAAAAA;
36         TCB_STACK[i][STACKSIZE-7] = 0xAAAAAAA;
37         TCB_STACK[i][STACKSIZE-8] = 0xAAAAAAA;
38         TCB_STACK[i][STACKSIZE-9] = 0xAAAAAAA;
39         TCB_STACK[i][STACKSIZE-10] = 0xAAAAAAA;
40         TCB_STACK[i][STACKSIZE-11] = 0xAAAAAAA;
41         TCB_STACK[i][STACKSIZE-12] = 0xAAAAAAA;
42         TCB_STACK[i][STACKSIZE-13] = 0xAAAAAAA;
43         TCB_STACK[i][STACKSIZE-14] = 0xAAAAAAA;
44         TCB_STACK[i][STACKSIZE-15] = 0xAAAAAAA;
45         TCB_STACK[i][STACKSIZE-16] = 0xAAAAAAA;
46     }

```

```

47
48     uint8_t osKernelAddThreads(void (*task0)(void), void (*task1)(void), void (*task2)(void)) {
49         __disable_irq();
50         tcbs[0].nextPt = &tcbs[1];
51         tcbs[1].nextPt = &tcbs[2];
52         tcbs[2].nextPt = &tcbs[0];
53         osKernelStackInit(0);
54         TCB_STACK[0][STACKSIZE - 2] = (int32_t)(task0);
55         osKernelStackInit(1);
56         TCB_STACK[1][STACKSIZE - 2] = (int32_t)(task1);
57         osKernelStackInit(2);
58         TCB_STACK[2][STACKSIZE - 2] = (int32_t)(task2);
59         currentPt = &tcbs[0];
60         __enable_irq();
61         return 1;
62     }
63
64     void osKernelInit(void) {
65         MILLIS_PRESCALER = (BUS_FREQ / 1000);
66     }
67
68     void osKernelLaunch(uint32_t quanta) {
69         SysTick->CTRL = SYSTICK_RST;
70         SysTick->VAL = 0;
71         SysTick->LOAD = (quanta * MILLIS_PRESCALER) - 1;
72         NVIC_SetPriority(SysTick_IRQn, 15);
73         SysTick->CTRL = CTRL_CLKSRC | CTRL_ENABLE;
74         SysTick->CTRL |= CTRL_TICKINT;
75         osSchedulerLaunch();
76     }
77 }
```

```

78     __attribute__((naked)) void SysTick_Handler(void) {
79         __asm("CPSID I");
80         __asm("PUSH {R4-R11}");
81         __asm("LDR R0, =currentPt");
82         __asm("LDR R1, [R0]");
83         __asm("STR SP, [R1]");
84         __asm("LDR R1, [R1, #4]");
85         __asm("STR R1, [R0]");
86         __asm("LDR SP, [R1]");
87         __asm("POP {R4-R11}");
88         __asm("CPSIE I");
89         __asm("BX LR");
90     }
91 }
```

```
92 void osSchedulerLaunch(void) {  
93     __asm("LDR R0, =currentPt");  
94     __asm("LDR R2, [R0]");  
95     __asm("LDR SP, [R2]");  
96     __asm("POP {R4-R11}");  
97     __asm("POP {R12}");  
98     __asm("POP {R0-R3}");  
99     __asm("ADD SP, SP, #4");  
100    __asm("POP {LR}");  
101    __asm("ADD SP, SP, #4");  
102    __asm("CPSIE I");  
103    __asm("BX LR");  
104}  
105
```

## Question 50

What is the significance of the TCB\_STACK array in the code?

- A) It stores the thread control blocks (TCBs) for each thread.
- B) It keeps track of the current thread's stack pointer.
- C) It holds the stack content for each thread.
- D) It contains the addresses of thread functions.

## Question 51

What does the `__asm("POP {R4-R11}")` instruction do in the `SysTick_Handler` function?

- A) Saves the values of registers R4 to R11 onto the stack.
- B) Loads the values of registers R4 to R11 from the stack.
- C) Enables the interrupt for registers R4 to R11.
- D) Restores the values of registers R4 to R11 from the stack.

### **Question 52**

What is the significance of the CTRL\_TICKINT constant in the osKernelLaunch function?

- A) It enables the SysTick timer.
- B) It sets the clock source for the SysTick timer.
- C) It configures the priority level of the SysTick interrupt.
- D) It enables the interrupt for the SysTick timer.

### **Question 53**

What is the purpose of the \_\_disable\_irq() and \_\_enable\_irq() functions in the osKernelAddThreads function?

- A) They disable and enable global interrupts, respectively.
- B) They disable and enable the SysTick interrupt, respectively.
- C) They disable and enable the scheduler, respectively.
- D) They disable and enable the thread context switching, respectively.

### **Question 54**

In the osSchedulerLaunch function, what is the purpose of the instruction `_asm("POP {R12}");?`

- A) It restores the value of register R12 from the stack.
- B) It saves the value of register R12 onto the stack.
- C) It skips the loading of register R12.
- D) It adds 4 to the stack pointer.

### **Question 55**

Which register(s) is/are saved onto the stack when the SysTick\_Handler function is executed?

- A) R0, R1, R2, R3, R12, LR, PC, PSR
- B) R4, R5, R6, R7, R8, R9, R10, R11
- C) R4, R5, R6, R7, R8, R9, R10, R11, R12
- D) R0, R1, R2, R3, R12, LR, PC, PSR, S0-S15

### **Question 56**

In the osKernelStackInit function, what is the purpose of setting the T-bit (bit 24) in the PSR register?

- A) It enables the Thumb mode for the processor.
- B) It disables the Thumb mode for the processor.
- C) It selects the interrupt mode for the processor.
- D) It sets the priority level for the processor.

### **Question 57**

What is the significance of the value 15 used in the NVIC\_SetPriority(SysTick\_IRQn, 15) statement?

- A) It sets the SysTick interrupt priority to the highest level.
- B) It sets the SysTick interrupt priority to the lowest level.
- C) It disables the SysTick interrupt.
- D) It enables the SysTick interrupt.

### **Question 58**

What is the purpose of the following code snippet?

```
/*Initial stack for thread0*/
osKernelStackInit(0);
/*Initial PC/
TCB_STACK[0][STACKSIZE - 2] = (int32_t)(task0);
```

- A) Initializes the stack for thread 0 with a specific function
- B) Initializes the link register for thread 0 with a specific function
- C) Initializes the stack for thread 0 with a specific address
- D) Initializes the program counter for thread 0 with a specific address

## Question 59

What is the purpose of the following code snippet?

```
/*Load Cortex-M SP from address equals R2, i.e. SP = currentPt->stackPt*/  
_asm("LDR SP,[R2]");
```

- A) Loads the value of the stack pointer (SP) from the address pointed to by R2
- B) Loads the value of the stack pointer (SP) from the address stored in R2
- C) Loads the value of R2 from the stack pointer (SP)
- D) Loads the value of R2 from the address pointed to by the stack pointer (SP)

FreeRTOS

## Question 60

Which of the following best describes FreeRTOS?

- A) Real-time operating system for desktop computers
- B) Open-source operating system for smartphones
- C) Real-time operating system for embedded systems
- D) Proprietary operating system for cloud servers

### **Question 61**

What is the purpose of a semaphore in FreeRTOS?

- A) To allocate memory dynamically
- B) To synchronize access to shared resources
- C) To schedule tasks based on priorities
- D) To handle inter-process communication

### **Question 62**

Which file contains the configuration settings for FreeRTOS?

- A) FreeRTOSConfig.h
- B) tasks.c
- C) queue.c
- D) portmacro.h

### **Question 63**

Which scheduling algorithm is supported by FreeRTOS?

- A) First-Come-First-Served (FCFS)
- B) Round Robin
- C) Shortest Job Next (SJN)
- D) Priority-based scheduling

### **Question 64**

Which function is used to create a new task in FreeRTOS?

- A) xTaskCreate()
- B) xTaskMake()
- C) xSemaphoreCreate()
- D) vTaskStartScheduler()

### **Question 65**

Which function is used to delay the execution of a task for a specified amount of time in FreeRTOS?

- A) xTaskCreate()
- B) vTaskDelay()
- C) xSemaphoreCreate()
- D) vTaskStartScheduler()

### **Question 66**

What is the function of the FreeRTOS idle task?

- A) To execute high-priority tasks
- B) To handle system initialization
- C) To handle memory allocation
- D) To execute when no other tasks are ready to run

```

1 const uint16_t *blue_led = (uint16_t *)BLUE;
2 const uint16_t *red_led = (uint16_t *)RED;
3 const uint16_t *orange_led = (uint16_t *)ORANGE;
4 const uint16_t *green_led = (uint16_t *)GREEN;
5
6 void SystemClock_Config(void);
7 static void MX_GPIO_Init(void);
8 static void MX_USART2_UART_Init(void);
9
10 void vLEDControllerTask(void *pvParameters);
11
12 typedef uint32_t TaskProfiler;
13
14 TaskProfiler blueTaskProfiler, redTaskProfiler, greenTaskProfiler, orangeTaskProfiler;
15
16 int __io_putchar(int ch)
17 {
18     HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFFFFF);
19     return ch;
20 }
21

```

```

22 int main(void)
23 {
24     HAL_Init();
25
26     SystemClock_Config();
27     MX_GPIO_Init();
28     MX_USART2_UART_Init();
29
30     xTaskCreate(vLEDControllerTask, "Blue LED Controller", 100, (void *)blue_led, 1, NULL);
31     xTaskCreate(vLEDControllerTask, "Red LED Controller", 100, (void *)red_led, 1, NULL);
32     xTaskCreate(vLEDControllerTask, "Green LED Controller", 100, (void *)green_led, 1, NULL);
33     xTaskCreate(vLEDControllerTask, "Orange LED Controller", 100, (void *)orange_led, 1, NULL);
34
35     vTaskStartScheduler();
36
37     while (1)
38     {
39     }
40 }
41
42 void vLEDControllerTask(void *pvParameters)
43 {
44     while (1)
45     {
46         HAL_GPIO_TogglePin(GPIOD, (uint16_t)pvParameters);
47         //blueTaskProfiler++;
48         for (volatile int i = 0; i < 100000; i++);
49     }
50 }
51

```

### **Question 67**

What does the xTaskCreate function do in the main function?

- A) Create a new task to initialize the system clock
- B) Create tasks to control different colored LEDs
- C) Create a task to configure the UART communication
- D) Create a task to toggle the GPIO pins

### **Question 68**

How many LED controller tasks are created in the main function?

- A) 1
- B) 2
- C) 3
- D) 4

### **Question 69**

What is the significance of the 100 parameter passed to xTaskCreate when creating the LED controller tasks?

- A) It specifies the stack size for each task
- B) It sets the priority level of each task
- C) It defines the maximum execution time for each task
- D) It determines the delay between task executions

```

void SenderTask(void *pvParameters)
{
    BaseType_t qState;
    /* Enter the blocked state for 200ms for space to become
     * available in the queue each time the queue is full
     */
    const TickType_t wait_time = pdMS_TO_TICKS(200);
    xQueue = xQueueCreate(3, 2);
    while (1)
    {
        qState = xQueueSend(xQueue, pvParameters, wait_time);
        if (qState != pdPASS)
        {
            /* Do something */
        }
        for (int i = 0; i < 100000; i++);
    }
}

void ReceiverTask(void *pvParameters)
{
    BaseType_t qState;
    while (1)
    {
        qState = xQueueReceive(xQueue, &xReceiveStruct, 0);
        if (qState == pdPASS)
        {
            if (xReceiveStruct.sDataSource == pressure_sensor)
            {
                printf("pressure sensor value = %d\r\n", xReceiveStruct.ucValue);
            }
            if (xReceiveStruct.sDataSource == humidity_sensor)
            {
                printf("humidity sensor value = %d\r\n", xReceiveStruct.ucValue);
            }
        }
    }
}

```

## Question 70

How many items can the queue hold in the code snippet?

- A) 1
- B) 2
- C) 3
- D) Unlimited

## Question 71

What happens if the xQueueSend function returns pdPASS in the SenderTask function?

- A) The task exits the loop and ends
- B) The task prints the sensor values
- C) The task proceeds to the next iteration of the loop
- D) The task blocks for 200ms

## Question 72

What does the pdMS\_TO\_TICKS(200) function call represent?

- A) Conversion of milliseconds to ticks
- B) Conversion of ticks to milliseconds
- C) Conversion of seconds to ticks
- D) Conversion of ticks to seconds

```
1 #define mainOnE_SHOT_TIMER          (pdMS_TO_TICKS(4000UL))
2 #define mainAUTO_RELOAD_TIMER_PERIOD (pdMS_TO_TICKS(500UL))
3
4 TimerHandle_t xAutoReloadTimer, xOneShotTimer;
5 BaseType_t xTimer1Started, xTimer2Started;
6
7 void prvOneShotTimerCallback(TimerHandle_t xTimer);
8 void prvAutoReloadTimerCallback(TimerHandle_t xTimer);
9
```

```

10 int main(void)
11 {
12     HAL_Init();
13     SystemClock_Config();
14     MX_GPIO_Init();
15     USART2_UART_TX_Init();
16     printf("system initializing.....\r\n");
17
18     xOneShotTimer = xTimerCreate("OneShot", mainOnE_SHOT_TIMER, pdFALSE, 0, prvOneShotTimerCallback);
19     xAutoReloadTimer = xTimerCreate("AutoReload", mainAUTO_RELOAD_TIMER_PERIOD, pdTRUE, 0, prvAutoReloadTimerCallback);
20     xTimer1Started = xTimerStart(xOneShotTimer, 0);
21     xTimer2Started = xTimerStart(xAutoReloadTimer, 0);
22
23     vTaskStartScheduler();
24
25     while (1)
26     {
27     }
28 }
29
30 void prvOneShotTimerCallback(TimerHandle_t xTimer)
31 {
32     static TickType_t xTimeNow;
33     xTimeNow = xTaskGetTickCount();
34     printf(" One Shot ticks so far %ld\r\n", xTimeNow);
35 }
36
37 uint32_t counter;
38 void prvAutoReloadTimerCallback(TimerHandle_t xTimer)
39 {
40     static TickType_t xTimeNow;
41     xTimeNow = xTaskGetTickCount();
42     printf("Auto reload ticks so far %ld\r\n", xTimeNow);
43
44     if ((counter++) == 20)
45     {
46         printf("timer stopped at %ld\r\n", xTimeNow);
47         xTimerStop(xAutoReloadTimer, 0);
48     }
49 }

```

## Question 73

What is the value of the mainOnE\_SHOT\_TIMER constant?

- A) 4000 milliseconds
- B) 500 milliseconds
- C) 4000 microseconds
- D) 500 microseconds

### **Question 74**

Which timer is set to autoreload mode?

- A) xOneShotTimer
- B) xAutoReloadTimer
- C) Both xOneShotTimer and xAutoReloadTimer
- D) None of the timers

### **Question 75**

What is the significance of the pdTRUE parameter when creating the xAutoReloadTimer?

- A) It specifies that the timer should be started immediately.
- B) It sets the timer to autoreload mode.
- C) It configures the timer to use a high-precision mode.
- D) It indicates that the timer should run on a different task.

### **Question 76**

What is the purpose of the xTimer1Started and xTimer2Started variables?

- A) They store the current tick count for the timers.
- B) They keep track of whether the timers have been started successfully.
- C) They control the frequency of the timer callbacks.
- D) They determine the duration of the timers.

### Question 77

What is the role of CMSIS-RTOS?

- A) They provide direct access to hardware peripherals.
- B) It serves as a wrapper around specific RTOS APIs.
- C) They are responsible for task scheduling and management.
- D) They enable interaction with general-purpose operating systems.

### Question 78

What is the purpose of CMSIS-RTOS APIs?

- A) To provide vendor-independent API standards
- B) To define hardware peripherals in a microcontroller
- C) To implement generic functions for RTOS programming
- D) To create a portable operating system interface

### Question 79

What is the benefit of using generic APIs in ARTOS programming?

- A) It allows RTOS code to be written once and run using different RTOS..

- B) It provides unique functionality not found in specific RTOS APIs.
- C) It simplifies the process of accessing hardware peripherals.
- D) It ensures compatibility with general-purpose operating systems.

### **Question 80**

What is the purpose of the osThreadNew API in CMSIS-RTOS?

- A) Creates a new mutex object
- B) Starts a new timer
- C) Creates a new task
- D) Allocates memory from a memory pool

### **Question 81**

Which CMSIS-RTOS API is used to create an event flag object?

- A) osMutexNew
- B) osThreadNew
- C) osEventFlagsNew
- D) osTimerNew

## Answer - Sheet

**Question 01-B**

**Question 04-B**

**Question 07-B**

**Question 10-A**

**Question 13-C**

**Question 16-B**

**Question 19-C**

**Question 22-C**

**Question 25-B**

**Question 28-D**

**Question 31-B**

**Question 34-B**

**Question 37-C**

**Question 40-B**

**Question 02-A**

**Question 05-B**

**Question 08-B**

**Question 11-B**

**Question 14-B**

**Question 17-A**

**Question 20-B**

**Question 23-C**

**Question 26-A**

**Question 29-D**

**Question 32-A**

**Question 35-A**

**Question 38-D**

**Question 41-C**

**Question 03-B**

**Question 06-C**

**Question 09-C**

**Question 12-B**

**Question 15-A**

**Question 18-A**

**Question 21-C**

**Question 24-B**

**Question 27-D**

**Question 30-A**

**Question 33-C**

**Question 36-B**

**Question 39-A**

**Question 42-C**

**Question 43-B**

**Question 46-D**

**Question 49-D**

**Question 52-D**

**Question 55-B**

**Question 58-D**

**Question 61-B**

**Question 64-A**

**Question 67-B**

**Question 70-C**

**Question 73-A**

**Question 76-B**

**Question 79-A**

**Question 44-A**

**Question 47-A**

**Question 50-C**

**Question 53-A**

**Question 56-A**

**Question 59-A**

**Question 62-A**

**Question 65-B**

**Question 68-D**

**Question 71-C**

**Question 74-B**

**Question 77-B**

**Question 80-C**

**Question 45-A**

**Question 48-C**

**Question 51-D**

**Question 54-A**

**Question 57-B**

**Question 60-C**

**Question 63-D**

**Question 66-D**

**Question 69-A**

**Question 72-A**

**Question 75-B**

**Question 78-A**

**Question 81-C**