* 1. <https://www.linkedin.com/pulse/good-embedded-systems-interview-questions-abd-el-ghafar-mohammed/>
  2. <https://www.javatpoint.com/linked-list-interview-questions>
  3. Memeory Barrrier
  4. How arm board is booted
     1. How init the memory
     2. What is the role of SPL/ MLO
     3. Who decompresses the Kernel image
     4. How device tree attached with kernel (How kernel know )

* 1. <https://www.techiedelight.com/bit-manipulation-interview-questions/> : Bit wise Question
  2. Implement stack using Linked list
     1. Push and pop operations
  3. Container of
  4. Offset of structure
  5. DMA
     1. Arch of DMA
  6. Kernel panic debugging
  7. Memory corruptions
  8. Queue Empty and full conditions
  9. Kernel space thread and what is the relation between user and kernel space thread
  10. What happened when a static variable is declared globally and also same variable is declared with a function. // no compiler error ,
  11. When to use mutex and spinlock , practical scenario
      1. Difference between both of them
  12. Memory allocation for 2D pointer
      1. Single pointer
      2. Double pointer
      3. Array of pointer
  13. Who decompress the kernel at boot level
  14. Spin lock:

An alternative behavior when the lock is contended is to put the current thread to sleep and wake it up when it becomes available. Then the processor can go off and execute other code. This incurs a bit of overhead—most notably the two context switches required to switch out of and back into the blocking thread, which is certainly a lot more code than the handful of lines used to implement a spin lock. Therefore, it is wise to hold spin locks for less than the duration of two context switches. Because most of us have better things to do than measure context switches, just try to hold the lock for as little time as possible[1](https://learning.oreilly.com/library/view/linux-kernel-development/9780768696974/ch10.html#ftn.ch10fn01) Later in this chapter we discuss *semaphores*, which provide a lock that makes the waiting thread sleep, rather than spin, when contended.

* 1. Demonstrate a hands-on technical background; excellent C programming and embedded, low level device driver development experience.
  2. Familiarity with computer system architecture, microprocessor, and microcontroller fundamentals (caches, buses, memory controllers, DMA, etc.)
  3. Experience with complex system-level debugging is invaluable.
  4. What is memory corruption's
  5. How DMA works in Linux
  6. How Cache memory work and architecture
  7. Why MMU and who initialize the MMU
  8. Process managements
  9. What are the area are shared between the treads.
  10. When to use semaphore and mutex
  11. what is he role of Register when kernel panic happened
  12. How to debug the kernel panic
  13. What is NULL pointer exceptions write a pseudo code
  14. Design your own lock , what the thing need to taken care and implement
  15. Design your own printf
  16. What is vitual memory and how it works
  17. Difference between spinlock and mutex
  18. Experience with complex system-level debugging is invaluable.
  19. What is memory corruption's
  20. How DMA works in Linux and its architecture
  21. How Cache memory work and architecture
  22. Why MMU and who initialize the MMU
  23. Process managements
  24. What are the area are shared between the treads.
  25. When to use semaphore and mutex
  26. what is the role of Register when kernel panic happened
  27. How to debug the kernel panic
  28. What is NULL pointer exceptions write a pseudo code
  29. Design your own lock , what the thing need to taken care and implement
  30. Design your own printf
  31. What is vitual memory and how it works
  32. Difference between spinlock and mutex
  33. Interrupt latency
  34. How Mutex is implemented in kernel
  35. How Linux system boot after power on.

**Q: Scheduling algorithm used, how you will decide sleep time -> On request signal, timer, button, interrupt**

* 1. Request Signal: In this approach, the program waits for a specific signal or event to occur before proceeding. For example, if the program is waiting for input from a user or a specific message from another component, it can enter a sleep state until the signal is received. The sleep time in this case is typically indefinite until the signal is received.
  2. Timer: A timer can be used to introduce a specific delay in the program execution. The program can set a timer for a certain duration, and during the sleep time, it can perform other tasks or wait until the timer expires. After the timer expires, the program can continue its execution.
  3. Button: If the program is running on a device with physical buttons or switches, it can enter a sleep state until a specific button or switch is pressed. The sleep time in this case depends on the user's interaction and can be indefinite until the button press occurs.
  4. Interrupt: Interrupts are hardware signals that can be triggered by external events or devices. When an interrupt occurs, the program execution is temporarily suspended, and control is transferred to the interrupt handler. The sleep time in this case is determined by the occurrence of the interrupt and can vary based on the specific event or device that triggers the interrupt.

The choice of which approach to use depends on the specific requirements and behavior expected from the program. It is essential to consider factors such as responsiveness, power consumption, real-time constraints, and overall system design when deciding on the appropriate sleep time mechanism.

Regenerate response

**Q.Blocking vs non-bloking design for receiver in Queue**

When designing a receiver in a queue, you can choose between a blocking or non-blocking design based on the desired behavior and requirements of your system. Here's an explanation of both approaches:

* 1. Blocking Design: In a blocking design, the receiver will block or wait until there is an item available in the queue. If the queue is empty when the receiver tries to receive an item, it will be suspended or put into a waiting state until an item is added to the queue. Once an item becomes available, the receiver resumes its execution and retrieves the item from the queue. This design ensures that the receiver does not consume any CPU resources while waiting for new items.

Pros:

* 1. Simple and straightforward implementation.
  2. Efficient resource utilization as the receiver is not actively consuming CPU resources during the wait time.
  3. Ensures synchronization between the sender and receiver, allowing for tight coupling.

Cons:

* 1. The receiver's execution may be delayed if the queue remains empty for an extended period.
  2. Difficult to handle scenarios where the receiver needs to perform other tasks while waiting for items.
  3. Non-blocking Design: In a non-blocking design, the receiver will check the queue for available items, but it will not block or wait if the queue is empty. Instead, it will return immediately, indicating that no items are currently available. The receiver can then continue with other tasks or periodically check the queue again until it finds an item. This design allows the receiver to perform other operations concurrently with the queue operations.

Pros:

* 1. Allows the receiver to perform other tasks while waiting for items, enabling better multitasking or responsiveness.
  2. Avoids potential delays in the receiver's execution caused by an empty queue.

Cons:

* 1. The receiver needs to implement additional logic to handle the case when the queue is empty.
  2. May consume more CPU resources as the receiver needs to continuously check the queue for items.

The choice between a blocking and non-blocking design depends on the specific requirements of your system. If the receiver's main purpose is to process items from the queue and no other significant tasks need to be performed concurrently, a blocking design can be simpler and more efficient. On the other hand, if the receiver needs to handle other operations or remain responsive while waiting for items, a non-blocking design would be more suitable.

* 1. **I2C clock stretching**

Hardware design pattern

Software design pattern

point to address and print

why we need unions

clock synchronization

reverse number

implement memcpy

Carrier 2nd:

Union structure applications

Linux boot sequence

OTA flow, Specially how it will download firmware?

Mutex psuedo code

Why MQTT uses TCP IP

Cross compilation debugging

Where configuration data will be stored

Exchanges between device and cloud. especially certificates -> Device cloud provisioning flow

basic queue example code

why semaphores are like that?

Serial port parameters

tcp vs udp

…

[19:41, 5/15/2023] MSFT Taral SHAH: Amazon 1st:

- Spin locks

- How you handle critical section and interrupts simultaneously? If you have incoming 5 IRQ with same priority what you will do?

- C code to reverse words? Follow up: Can you do it in place?

- Internal implementation of condition variables?

- How virtual memory works?

- Reverse words of string.

- Have you ever worked on code optimization

[19:41, 5/15/2023] MSFT Taral SHAH: Amazon 2nd:

- Most challenging problem - Detailed questions into this

- Who checks permissions of user in linux

- What happens when we reboot the device

- What happens when we write data to specific memory location?

- Code for OTA

- Did you ever worked upon code optimization at compiler stage?

- what are mutex, semaphore, spin locks

Amazon 3rd:

- most difficult situation

- Critical feedback you got

- Learned and simplified

- Curious and provided value addition to them which customer didn't had idea of

- When we missed timeline and managed.

[19:41, 5/15/2023] MSFT Taral SHAH: Amazon 4th:

- Difference between Embedded linux and RTOS

- Cache coherency problem

- How to synchronize two cpu threads? One is running Linux, 2nd is running RTOS

- 4 cpu cores are synchronized properly, then how many mutex we need for it? how many spind locks we need?

- What are run loop queues?

- If we have 4 cpus running individual threads. Each one keeps increamenting i. How we can ensure that it exactly increases it till 1000? Different cores might be running at different speed.

- Where this variable will be shared?

- Code to find numeber of ways to split good substrings.

- Run loop Queues

Amazon 5th:

- What is your weakness? Sometimes taking more time. Detailed situation disucssion for this.

- What was situation where you missed deadline?

- Timer library that is scalable

- Producer consumer problem.