Q. Explain task synchronisation

Synchronization and messaging provides the necessary communication between tasks in one system to tasks in another system. The event flag is used to synchronize internal activities while message queues and mailboxes are used to send text messages between systems. Common data areas utilize semaphores. Below are top 3 messaging and synchronization techniques :-

*Message Queues :-

Message queues are data buffers with a finite amount of entries. Each entry can contain data of a certain size (eg. 32 bits). Message queues can be used for passing data between tasks and between interrupt service routines and tasks. They are implemented as thread-safe FIFO buffers. Specific actions are defined in the RTOS in case a task tries to write to a full message queue or tries to read an empty one.

The common operations that can be performed on message queues are :-

- Create / delete a message queue.
- Get the number of messages currently stored in the queue.
- put a message into the queue.
- Get a message from the queue.

*Mailboxes :-

A mailbox can store a single data of specific size (eg. 32 bit variable) and can be implemented as a single-entry queue. A single mailbox can be accessed by many tasks. In some RTOS distributions, mailboxes can have more than one entry, which makes them very similar to what we described as a message queue.

Typical operation that can be performed on mailbox are :-

- Create / delete a mailbox.
- Write to a mailbox
- Read a mailbox.

*Semaphores :-

Semaphores are independent kernal objects, which provide a flagging mechanism that is generally used to control access to a resource. There are broadly two types: binary semaphores (that just have two states) and counting semaphores (that have an arbitrary number of states). Some processors support (atomic) instructions that facilitate the easy implementation of binary semaphores. Binary semaphores may also be viewed as counting semaphores with a count limit of 1.

Semaphore is a signalling mechanism. It is available in all real-time operating systems with some subtle differences in its implementation. Semaphores are used for synchronization (between tasks or between tasks and Interrupts) and managing allocation and access to shared resources.

*mutexes :-

mutual exclusion Semaphores - mutexes - are independent kernel objects, which behave in a very similar way to normal binary semaphores. They are slightly more complex and incorporate the concept of temporary ownership (of the resource, access to which is being controlled). If a task obtains a mutex, only that same task can release it again - the mutex (and, hence, the resource) is temporarily owned by the task.

Q. Explain task Scheduling and communication

Task Scheduling in embedded systems involves managing and coordinating various tasks or processes that need to be executed within the system. These tasks could range from handling sensor data, processing user inputs, controlling actuators, managing communication protocols, and performing other system specific operations. The primary goal of task scheduling is to efficiently utilize the limited computational resources available in embedded systems, such as the CPU, memory and peripherals.

Embedded systems often operate in real-time environments, where tasks must meet specific timing constraints and deadlines. Therefore, the scheduler must ensure that critical tasks are prioritized and executed within their specified timeframes. Scheduling algorithms play a vital role in determining the order in which tasks are processed and how the system resources are allocated. Some common scheduling algorithms used in embedded systems include :-

* Round-robin scheduling :- This algorithm assigns an equal amount of the time to each task in a circular manner, ensuring fair execution of tasks.

* Priority-based scheduling :- Tasks are assigned priorities, and the scheduler executes the higher priority tasks before the lower priority ones. This is crucial for ensuring that time-critical tasks are processed on time.

* Preemptive scheduling :- This algorithm allows higher priority tasks to preempt or interrupt lower priority tasks. It ensures that time-critical operations can take precedence over less critical ones, improving system responsiveness.

Communication in embedded systems refers to the transfer of data and information between various components, such as microcontrollers, sensors, actuators and other embedded devices. Efficient communication is critical for achieving seamless interaction and coordination among different parts of the embedded system. we can say that the task communication implemented in real-time embedded systems in based on one or a combination of the following mechanisms :-

- memory sharing
- Signaling mechanism
- message passing

*Memory Sharing :-

The first thing that comes to mind as a mechanism for passing information between different tasks is using a shared memory location, It is important that the shared memory is protected either by a mutex or semaphore (See RTOS mutex and Semaphore Basics). A very basic example of using shared memory location is a global variable. Although there is nothing stopping us from using such a variable, it is not recommended as there are far more sophisticated ways available as a means of communication between tasks.

*Signalling mechanisms :-

Signalling mechanisms are a basic form of communication They indicate the occurence of an event and can be used for synchronization purposes. we will present two additional Signalling mechanisms that have wide use in real-time operating systems :- event flags and task flags. As there are no strict naming conventions, you may find these mechanisms under slightly different names in different RTOS distributions, but the function they serve is pretty much the same.

• Event Flags :-

Event flags are bits used to encode specific information. They are used for synchronizing tasks and communication. The grouping of individual event flag is called an event group or a signal. Events flags can be used by tasks and by interrupt service routines (ISR). A single event flag (or a group) can be accessed by many different tasks. The most common operations that can be performed on event flags are :-

- Create/delete event flags.
- Set/clear event flags.
- Read a flag's value.
- Wait on a flag to take a specific value.

• Task flags :-

Task flags are a special form of event flags. While event flags can be accessed by all tasks, the task flags are used for notifications to a single receiving task. The most common operations that can be performed on task flags are :-

- Set/clear flags of a specific task.
- Wait on a flag to take a specific value.

* Message passing :-

we can generally define two types of message passing :-

• Direct message passing :- The sender and the receiver of the messages are explicity defined. As an example, the popular FreeRTOS has Stream & Message Buffers as primitives for direct message passing between a single writer task and a single reader task.

- Indirect message passing :- The messages are placed in structures such as message queues or mailboxes and multiple tasks have read/write access.

Here are some examples of task scheduling and communication in embedded systems :

- A car's engine control unit (ECU) uses task scheduling to manage the execution of various tasks, such as controlling the fuel injection system, the ignition system, and the emissions control system.
- A smartphone uses task scheduling to manage the execution of multiple applications at the same time.
- A robot uses task communication to allow different parts of the robot to coordinate their movements.
- A smart home device uses task communication to allow different devices in the home to interact with each other.