24EC504 – Unit III – Part 2: Inter-Task Communication and the MQTT Protocol

1. Inter-Task Communication and Synchronization

There are three broad paradigms:

- **Task-owned facilities:** attributes the RTOS gives a task for input, the classic example being a **signal**.
- **Kernel objects:** stand-alone facilities provided by the RTOS: event flags, mailboxes, queues/pipes, semaphores and mutexes.
- **Message passing:** the RTOS lets you create message objects that can be sent from one task to another or to several tasks; fundamental to message-passing RTOS designs.

1.1 Shared Variables / Shared Memory

Multiple tasks access the same memory location: one writes, another reads. Access must be synchronized with locks.

- *Advantages:* simple, fast, minimal memory overhead.
- *Limitations:* risk of data corruption, needs a locking mechanism, poor for large transfers.

1.2 Signals

Bit flags that notify a task of an event. Any task can set signal bits, but only the owner task reads them; reading usually clears the signal.

- *Advantages:* very fast, low memory, suitable for event notification.
- *Applications:* task synchronization, interrupt notification.

1.3 Mutex (Mutual Exclusion)

Protects a shared resource. Only one task can own it at a time, and the same task that took it must release it, preventing simultaneous access.

- *Advantages:* prevents race conditions, keeps data intact, safer than a binary semaphore for resource protection.
- *Applications:* shared memory, shared peripherals, file access.

1.4 Event Flag Groups

A collection of event flags shared among tasks. Any task can set or clear flags, and tasks wait until a required combination occurs.

- *Advantages:* handles several events together, supports blocking waits, flexible synchronization.

- *Applications:* coordinating multiple tasks, event-driven systems.

1.5 Semaphores

A synchronization object controlling access to a shared resource: a task obtains it, uses the resource, then releases it.

- **Binary semaphore:** value 0/1, used for signalling.
- **Counting semaphore:** tracks multiple available instances of a resource.
- *Advantages:* prevents resource conflicts, supports synchronization, allows blocking until the resource is free.

1.6 Mailboxes

Transfer a single message between tasks. The sender places a message, the receiver reads it, and the mailbox is empty afterwards.

- *Advantages:* simple, suits small messages, supports blocking send/receive.
- *Applications:* command passing, status messages.

1.7 Queues

A message buffer that stores several messages in FIFO order: added at the rear, removed from the front.

- *Features:* FIFO, multiple messages, blocking send/receive, and message priority through "jamming" (inserting an urgent message at the front).
- *Applications:* sensor data collection, producer–consumer systems.

1.8 Summary of ITC Mechanisms

Mechanism	Purpose	Best used for
Shared memory	Data sharing	Fast communication
Signals	Event notification	Interrupt handling
Event flags	Multiple-event synchronization	Task coordination
Semaphore	Resource control	Shared resources
Mailbox	Single message transfer	Commands and notifications
Queue	Multiple message transfer	Sensor/communication data
Mutex	Mutual exclusion	Protecting critical resources

2. MQTT Protocol

2.1 Principle: Publish/Subscribe

In traditional networking a client requests data directly from a server. MQTT instead decouples the sender (publisher) from the receiver (subscriber) with a **message broker**, which filters incoming messages and distributes them to the right subscribers.

Three kinds of decoupling provided by the broker:

- **Space decoupling:** publisher and subscriber do not know each other's network location (no IP addresses or ports exchanged).
- **Time decoupling:** they need not be connected at the same time.
- **Synchronization decoupling:** publishing and subscribing do not block each other; a subscriber need not wait for the publisher.

2.2 MQTT Components

- **MQTT client:** any device, from a server to a microcontroller, running an MQTT library. It is a publisher when sending and a subscriber when receiving.
- **MQTT broker (server):** the backend that coordinates messages. Responsibilities: receive and filter messages; identify subscribed clients and deliver to them; authenticate and authorize clients; optionally pass messages on for analysis; handle missed messages and client sessions; reject messages that break the protocol specification.
- **MQTT connection:** a client sends a **CONNECT** message; the broker confirms with **CONNACK**. Both need a TCP/IP stack (WebSockets also supported). Clients never connect to each other directly, only to the broker.

2.3 How MQTT Works

1. A client connects to the broker.
2. Once connected, it can publish, subscribe to topics, or both.
3. When the broker receives a message, it forwards it to all subscribers interested in that topic.

2.4 Topics

A topic is the keyword (or hierarchy) the broker uses to filter and route messages. Topics are hierarchical, like folders and files. A smart-home example:

- `ourhome/groundfloor/livingroom/light`
- `ourhome/firstfloor/kitchen/temperature`

2.5 Publish and Subscribe

- **Publish:** a message contains a topic and a payload in bytes; the client chooses the format (text, binary, XML, JSON). A smart lamp may publish on `livingroom/light`.

- **Subscribe:** a client sends a SUBSCRIBE message with a unique packet identifier and a list of subscriptions. A smart-home phone app might subscribe to the wildcard `+/light` and count every "on" message.

2.6 Quality of Service (QoS)

QoS	Guarantee	Typical use
0	At most once: fire and forget, no acknowledgement	Frequent, non-critical sensor readings
1	At least once: acknowledged, duplicates possible	Commands where delivery matters and duplicates are tolerable
2	Exactly once: no duplicates, highest overhead	Billing, critical control commands

2.7 Other Useful Concepts

- **Retained message:** the broker stores the last message on a topic and delivers it to any new subscriber immediately.
- **Last Will and Testament (LWT):** a message the broker publishes automatically if a client disconnects unexpectedly, signalling device failure or offline status.
- **Keep-alive:** a periodic PING lets the broker detect a silent client.
- MQTT is lightweight (small header), works over low-bandwidth/high-latency links and uses little power, which makes it the most widely used IoT telemetry protocol compared with heavier ones such as HTTP.