Kernel objects are special constructs that are the building blocks for application development for real-time embedded systems.Counting Semaphores o A mutual exclusion (mutex) semaphore is a special binary semaphore that supports ownership, task deletion safety, and one or more protocols for avoiding problems inherent to mutual exclusion.Typically, the more deadlines to be met-and the shorter the time between them-the faster the system's CPU must be. o Although underlying hardware can dictate a system's processing power, its software can also contribute to system performance.To meet total system requirements, designers must understand both the static and dynamic memory consumption of the RTOS and the application that will run on it. Compactness o Because RTOSes can be used in a wide variety of embedded systems, they must be able to scale up or down to meet application-specific requirements.As mentioned earlier, developers decompose applications into multiple concurrent tasks to optimize the handling of inputs and outputs within set time constraints.Semaphores o semaphore (sometimes called a semaphore token) is a kernel object that one or more threads of execution can acquire or release for the purposes of synchronization or mutual exclusion.The most common RTOS kernel objects are: o Tasks are concurrent and independent threads of execution that can compete for CPU execution time.Some of the more common attributes are: o Reliability o Predictability o Performance o Compactness o Scalability Key Characteristics of an RTOS o Embedded systems must be reliable.o Depending on how much functionality is required, an RTOS should be capable of adding or deleting modular components, including file systems and protocol stacks.Defining Semaphores o When a semaphore is first created, the kernel assigns to it an associated semaphore control block (SCB), a unique ID, a value (binary or a count), and a task-waiting list.o Semaphores are token-like objects that can be incremented or decremented by tasks for synchronization or mutual exclusion.The term deterministic describes RTOSes with predictable behavior, in which the completion of operating system calls occurs within known timeframes.o Kernels are the core module of every RTOS and typically contain kernel objects, services, and scheduler.Defining a Task A task is schedulable, and the task is able to compete for execution time on a system, based on a predefined scheduling algorithm.Task States and Scheduling o blocked state-the task has requested a resource that is not available, has requested to wait until some event occurs, or has delayed itself for some duration.o Multiple concurrent threads of execution within an application must be able to synchronize their execution and coordinate mutually exclusive access to shared resources.Defining Semaphores o The kernel tracks the number of times a semaphore has been acquired or released by maintaining a token count, which is initialized to a value when the semaphore is created.These services comprise sets of API calls that can be used to perform operations on kernel objects or can be used in general to facilitate timer management, interrupt handling, device I/O, and memory management.However, this scheme is inappropriate for real-time embedded applications, which generally handle multiple inputs and outputs within tight time constraints.Tasks o Concurrent design requires developers to decompose an application into small, schedulable, and sequential program units.o Most RTOS kernels provide task objects and task management services to facilitate designing concurrency within an application.Defining a Task A task is defined by its distinct set of parameters and supporting data structures.o The reserved priority levels refer to the priorities used internally by the RTOS for its system tasks.Defining a Task Generally three main states are used in most typical preemptive-scheduling kernels, including: o ready state-the task is ready to run but cannot because a higher priority task is executing.o As a task acquires the semaphore, the token count is decremented; as a task releases the semaphore, the count is incremented.o Message Queues are buffer-like data structures that can be used for synchronization, mutual exclusion, and data exchange by passing messages between tasks.o Developers can write simple benchmark programs to validate the determinism of an RTOS.Scalability Some points to remember include the following: o RTOSes are best suited for real-time, application-specific embedded systems; GPOSes are typically used for general-purpose systems.o RTOSes are programs that schedule execution in a timely manner, manage system resources, and provide a consistent foundation for developing application code.o RTOSes for real-time embedded systems should be reliable, predictable, high performance, compact, and scalable.To sum up Simple software applications are typically designed to run sequentially , one instruction at a time, in a pre-determined chain of instructions.Tasks A task is an independent thread of execution that can compete with other concurrent tasks for processor execution time.o When the kernel first starts, it creates its own set of system tasks and allocates the appropriate priority for each from a set of reserved priority levels.When creating a counting semaphore, assign the semaphore a count that denotes the number of semaphore tokens it has initially.Objects Along with objects, most kernels provide services that help developers create applications for real-time embedded systems.For example, a digital solar-powered calculator might reset itself if it does not get enough light.o common way that developers categorize highly reliable systems is by quantifying their downtime per year.Predictability o This requirement dictates that a system must perform fast enough to fulfill its timing requirements.Performance o Application design constraints and cost constraints help determine how compact an embedded system can be. For example, a cell phone clearly must be small, portable, and low cost.If an RTOS scales well, the same RTOS can be used in both projects, instead of two different RTOSes, which saves time and money.When done correctly, concurrent design allows system multitasking to meet performance and timing requirements for a real-time system.Specifically, upon creation, each task has an associated name, a unique ID, a priority, a task control block (TCB), a stack, and a task routine.o To address these requirements, RTOS kernels provide a semaphore object and associated semaphore management services.A requesting task, therefore, cannot acquire the semaphore, and the task blocks if it chooses to wait for the semaphore to become available.Note that when a binary semaphore is first created, it can be initialized to either available or unavailable (1 or 0, respectively).Counting Semaphores o If the initial count is greater than 0, the semaphore is created in the available state, and the number of tokens it has equals its count.This feature allows any task to release a counting semaphore token.o A mutex is initially created in the unlocked state, in which it can be acquired by a task.Services An application's requirements define the requirements of its underlying RTOS.On the other hand, a telecom switch cannot reset during operation without incurring high associated costs for down time.Benchmarks are written by producing timestamps when a system call starts and when it completes.Real-time embedded software applications must be designed for concurrency.o A semaphore is like a key that allows a task to carry out some operation or to access a resource.In this sense, acquiring a semaphore is like acquiring the duplicate of a key from an apartment manager when the apartment manager runs out of duplicates, the manager can give out no more keys.o Likewise, when a semaphore s limit is reached, it can no longer be acquired until someone gives a key back or releases the semaphore.o If the token count reaches 0, the semaphore has no tokens left.o These blocked tasks are kept in the task-waiting list in either first in/first out (FIFO) order or highest priority first order.Defining Semaphores o A binary semaphore can have a value of either 0 or 1.Binary Semaphores o Binary semaphores are treated as global resources, which means they are shared among all tasks that need them.Each release operation increments the count by one, even if the task making this call did not acquire a token in the first place.An unbounded count allows the counting semaphore to count beyond the initial count.Depending on the application, the system might need to operate for long periods without human intervention.Reliability o The RTOS needs to be predictable to a certain degree.The result is based on timed responses to specific RTOS calls.In a good deterministic RTOS, the variance of the response times for each type of system call is very small.o An application should avoid using these priority levels for its tasks because running application tasks at such level may affect the overall system performance or behavior.o A single semaphore can be acquired a finite number of times.When a binary semaphore's value is 0, the semaphore is considered unavailable (or empty); when the value is 1, the binary semaphore is considered available (or full).o A counting semaphore uses a count to allow it to be acquired or released multiple times.o Counting semaphores are also global resources that can be shared by all tasks that need them.o Some implementations of counting semaphores might allow the count to be bounded.A bounded count is a count in which the initial count set for the counting semaphore, determined when the semaphore was first created.o The states of a mutex are unlocked or locked (0 or 1, respectively).o Sometimes developers measure RTOS performance on a call-by-call basis.These design requirements limit system memory, which in turn limits the size of the application and operating system.o The RTOS clearly must be small and .efficient.Together, these components make up what is known as the task object