Mutexes and semaphores are synchronization primitives used to manage access to shared resources in concurrent programming. While both aim to prevent race conditions and ensure data consistency, they differ in their functionality and use cases. Mutexes provide mutual exclusion, allowing only one thread to access a resource at a time, while semaphores can control access for a specified number of threads simultaneously.

Mutex (Mutual Exclusion):

- A mutex is a locking mechanism that ensures exclusive access to a shared resource. Only one thread can "lock" (acquire) the mutex at a time.
- When a thread acquires a mutex, other threads attempting to acquire the same mutex will be blocked until the first thread releases it.
- Mutexes are typically used for situations where only one thread should be allowed to modify a shared resource at a given time, preventing data corruption.
- They have the concept of ownership, meaning the thread that locks the mutex should be the one to unlock it. Violating this can lead to synchronization issues.
- A mutex is essentially a binary semaphore, allowing only one thread to access the resource at a time.

Semaphore:

- A semaphore is a signaling mechanism that allows controlling access to a shared resource for a specified number of threads simultaneously.
- It acts as a counter that represents the number of available resources or the extent to which a resource is available.
- When a thread needs a resource, it checks the semaphore's value. If it's greater than zero, the thread acquires the resource (decrementing the counter) and proceeds. If it's zero, the thread waits until the resource becomes available (counter becomes greater than zero).
- Semaphores are suitable for managing access to a limited number of resources, like a connection pool or a limited number of printer slots.
- There are two main types of semaphores:
  - **Binary Semaphore:** Similar to a mutex, allowing only one thread to access a resource at a time.
  - **Counting Semaphore:** Allows a specified number of threads to access a resource concurrently.
- Unlike mutexes, semaphores don't necessarily have the ownership concept. Any thread can increment or decrement the semaphore's value, even if it didn't acquire it originally.

Key Differences Summarized:

- **Access Control:**

  Mutexes provide exclusive access (one thread at a time), while semaphores can control access for multiple threads up to a limit.

- **Ownership:**

  Mutexes have an ownership concept (the locking thread should be the unlocking thread), while semaphores generally don't.

- **Use Cases:**
  Mutexes are best for protecting critical sections where only one thread can execute at a time. Semaphores are more suitable for managing access to a limited number of resources.