1. In single-processor systems, using spinlocks can cause problems as they rely on a different process to change the state of the program so that the process waiting in the spinlock can make progress. However, if the process holding the spinlock never releases the processor, other processes are not given the chance to modify the program state and release the first process. This issue is not present in multiprocessor systems as multiple processes can run simultaneously on different processors, allowing for the program state to be modified and for the spinlock to be released. It's important to consider the architecture of the system before implementing spinlocks in order to ensure that they are being used in an appropriate manner.

2. A race condition can occur in this situation. If multiple users call the bid function at the same or similar time with different bid amounts, it is possible that two or more bids might be accepted simultaneously, resulting in a situation where the highest bid is not accurately maintained.

Suppose A and B both want to bid on an item in the online auction system. They both call the bid function at the same time with different bid amounts. The current highest bid is $100, and User A bids $150, while User B bids $125. Without proper synchronization, both bids could be accepted simultaneously.

- A's bid is checked first, and since $150 is higher than $100, the highest bid is set to $150.
- Before A's bid is fully processed, B's bid is checked, and since $125 is higher than $100 (but not higher than $150), the highest bid is set to $125.
- A's bid is then fully processed, but since the highest bid is already set to $125, their bid is not accepted.

As a result, the highest bid is not accurately maintained, and A might not have had a fair chance to bid on the item. Proper synchronization mechanisms could prevent this race condition from occurring.

3. Semaphores and Monitors are synchronization mechanisms used by threads to control access to shared resources. Semaphores provide a low-level primitive that can be used for enforcing mutual exclusion and signaling events between threads, but they don't have built-in protection of shared data. Monitors, on the other hand, provide a higher-level abstraction that encapsulates shared data and the operations that can be performed on it. Monitors have built-in protection of shared data, and only one thread can have exclusive access to the monitor at a time. Monitors also have built-in condition variables that allow threads to wait for specific conditions to be met.

It is the programmer's responsibility to protect shared data when using semaphores, while monitors provide built-in protection. If shared data needs to be protected and operations on that data need to be atomic, a monitor may be a better choice. If lower-level control over synchronization is required, such as signaling between threads, semaphores may be more suitable.

4. The three classical problems of synchronization are the Producer-Consumer problem, the Reader-Writer problem, and the Dining Philosophers problem.

The Producer-Consumer problem involves two types of threads: producers and consumers. Producers generate items and place them in a shared buffer, while consumers remove items from the buffer and process them. The problem is to ensure that the producers and consumers do not access the buffer simultaneously and that the buffer is not accessed when it is empty or full.

The Reader-Writer problem involves multiple threads that can either read or write a shared resource. Multiple readers can access the resource simultaneously, but when a writer accesses the resource, no other reader or writer can access it. The problem is to ensure that a writer does not starve and that readers do not prevent a writer from accessing the resource.

The Dining Philosophers problem involves multiple philosophers sitting around a table, with a chopstick between each pair of philosophers. Each philosopher needs two chopsticks to eat. The problem is to design a protocol that allows the philosophers to eat without deadlocking or starving. If each philosopher takes the chopstick to their left and then waits for the chopstick to their right, a deadlock can occur if all philosophers pick up their left chopstick at the same time. If each philosopher waits for both chopsticks to become available, starvation can occur if one philosopher is always waiting for the chopsticks.