1. We call the situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place condition as race condition. Now, we could assume the value of highestBid = 100; if there are two person call the bid function, one is bit(200) and another one is bit(300), and both of them are greater than 100, both of then run after the if statement and before the highestBid = amount. Then, the order would be these two situations:
   a) highestBid = 200 then highestBid =300, result highestBid = 300
   b) highestBid = 300 then highestBid =200, result highestBid = 200

   We can do this to prevent the race condition from occurring.
   ```
   void bid (double amount) {
       acquire();
       if (amount > highestBid)
           highestBid = amount;
       release();
   }
   ```

2.
   ```
   mutex->available = 0;
   void acquire (lock *mutex){
       while(compare_and_swap (&mutex->available, 0, 1) != 0) // busy wait
       return;
   }
   void release (lock *mutex){
       mutex->available = 0;
       return;
   }
   ```

3. a) The lock is to be held for a short duration.
   Spinlock. On multicore systems, spinlocks are the preferable choice for locking if a lock is to be held for a short duration.
   b) The lock is to be held for a long duration.
   Mutex lock. It allows the other processing core to schedule another process while the locked process waits.
   c) A thread may be put to sleep while holding the lock.
   Mutex lock. If the process holding the lock sleeps, there's no point in waiting processes to keep spinning until the current process wakes up.

4. a) Race condition: 'number of processes'
   b) In allocate_process(), we can add '**acquire()**' before 'int new pid', and at the end of the function, add the '**release()**'. And in release_process(), put '—number_of_processes' between **acquire()** and **release().**

5. If wait operation is performed, then the thread will be blocked. The signal() operation associated with the monitor is not persistent. Even without waiting threads, each singal() results in an increase in the value of the signal.