- **data race**
  - **竞争与协作**
    - 互斥的概念
      - 由于多线程执行操作共享变量的这段代码可能会导致竞争状态，因此我们将此段代码称为临界区（critical section），它是访问共享资源的代码片段，一定不能给多线程同时执行
      - 我们希望临界区的代码是「互斥（mutualexclusion）」的，在同一时间保证一个线程在临界区执行，其他线程应该被阻止在临界区外
    - 同步的概念
      - 并发进程/线程在一些关键点可能需要互相等待与互通信息，这种相互制约的等待与互通信息称为进程/线程同步
    - 同步和互斥的区分
      - 同步就好比「操作A应该在操作B之前」、「操作C必须在操作完A和B完成后」
      - 互斥就好比「A、B不能同一时间执行」
  - **互斥与同步的实现**
    - 锁
      - 任何想进入临界区的线程必须先执行加锁操作。如果加锁成功则进入临界区，完成对临界区的资源后进行解锁操作
      - 实现区分
        - 忙等待锁（自旋锁，一直请求直到获取锁）
        - 无忙等待锁（尝试获取锁，获取不到可以做其他事）
    - 信号量
      - 信号量是操作系统的一种协调共享资源访问的方法
      - P、V操作
        - 通常「信号量表示资源的数量」，对应一个整型变量（sem）
        - P操作：将「sem」减「1」，如果「sem<0」,则进程/线程进入阻塞等待，否则继续，表名P操作可能会阻塞
        - V操作：将「sem」加「1」，如果「sem<=0」则唤醒等待中的进程/线程，表名V操作不会阻塞
        - P操作用于在进入临界区之前，V操作是用于离开临界区之后，两个操作必须「成对出现」
      - P、V使用
        - 实现临界区的互斥访问控制
          - 将「sem」设置为「1」
          - 「sem」==1，表示没有线程进入临界区
          - 「sem」==0，表示有一个线程进入临界区
          - 「sem」==-1，表示一个线程进入临界区，另一个线程等待进入
        - 实现线程间的事件同步
          - 将「sem」设置为「0」
          - 先执行P操作「sem」==-1，等待V操作
          - V操作执行完成后唤醒阻塞的P后，又执行P等待执行结果
          - 唤醒的P执行完成后，执行V通知完成
    - 生产者-消费者问题
      - 「生产者」在生成数据后，放到一个buffer中
        「消费者」从buffer中取出数据处理
        在「同一时刻」只能有一个「生产者」或「消费者」访问
      - 任何时刻只能有一个线程操作缓冲区，说明操作缓冲区是临界代码，需要互斥；
      - 缓冲区空时，消费者必须等待生产者生成数据；
        缓冲区满时，生产者必须等待消费者取出数据。
        说明生产者和消费者需要同步。
      - 互斥信号量 mutex：用于互斥访问缓冲区，初始化值为 1；
        资源信号量 fullBuffers：用于消费者询问缓冲区是否有数据，有数据则读取数据，初始化值为 0（表明缓冲区一开始为空）；
        资源信号量 emptyBuffers：用于生产者询问缓冲区是否有空位，有空位则生成数据，初始化值为 n（缓冲区大小）；
        - 消费者先执行p(fullBuffers)如果有内容可以消费，则在执行p(mutex)，消费完成后v(emptyBuffers)，v(mutex)
          生产者执行p(emptyBuffers)如果有空位执行p(mutex)放到临界区后v(fullBuffers)，v((mutex))
  - **经典同步问题**
    - 哲学家就餐问题（死锁问题）
      - 信号量简单的pv操作——造成死锁
      - 信号量+互斥信号——同时只能有一个进餐
      - 互斥信号量+奇偶分离（奇数先拿左在拿右、偶数先拿右再拿左）——不会出现死锁且同时有两人进餐
      - 用数组记录状态，相邻的没有进餐才可以进入就餐状态——不会出现死锁且同时又两人进餐
    - 读者-写者问题（读写锁问题）
      - 使用信号量方式解决——如果读者/写着不断进入，写着/读者 处于饥饿状态
      - 增加flag标志位，划清水位线后来的需要排队，实现公平策略