1、线程与进程的区别联系
2、进程通信方式有哪些？
3、同步的方式有哪些？
4、ThreadLocal 与其它同步机制的比较
5、进程死锁的条件

第一题：
（1）线程是进程的一个实体，一个进程可以拥有多个线程，多个线程也可以并发执行。一个没有线程的进程也可以看做是单线程的，同样线程也经常被看做是一种轻量级的进程。并且进程可以不依赖于线程而单独存在，而线程则不然。
（2）进程是并发程序在一个数据集合上的一次执行过程，进程是系统进行资源分配和调度的独立单位，线程是进程的实体，它是比进程更小的能够独立执行的基本单元，线程自己不拥有任何系统资源，但是它可以访问其隶属进程的全部资源。
（3） 与进程的控制表 PCB 相似，线程也有自己的控制表 TCB，但是 TCB 中所保存的线程状态比 PCB 表少得多。
进程的作用与定义：是为了提高 CPU 的执行效率，为了避免因等待而造成 CPU 空转以及其他计算机硬件资源的浪费而提出来的。
线程的引入：例如，有一个 Web 服务器要进程的方式并发地处理来自不同用户的网页访问请求的话，可以创建父进程和多个子进程的方式来进行处理，但是创建一个进程要花费较大的系统开销和占用较多的资源。除外，这些不同的用户子进程在执行的时候涉及到进程上下文切换，上下文切换是一个复杂的过程。所以，为了减少进程切换和创建的开销，提高执行效率和节省资源，人们在操作系统中引入了"线程（thread）"的概念。

第二题：
进程间通讯的方式：

- 管道中还有命名管道和非命名管道之分，非命名管道只能用于父子进程通讯，命名管道可用于非父子进程，命名管道就是 FIFO，管道是先进先出的通讯方式。FIFO 是一种先进先出的队列。它类似于一个管道，只允许数据的单向流动。每个 FIFO 都有一个名字，允许不相关的进程访问同一个 FIFO，因此也成为命名管。
- 消息队列：是用于两个进程之间的通讯，首先在一个进程中创建一个消息队列，然后再往消息队列中写数据，而另一个进程则从那个消息队列中取数据。需要注意的是，消息队列是用创建文件的方式建立的，如果一个进程向某个消息队列中写入了数据之后，另一个进程并没有取出数据，即使向消息队列中写数据的进程已经结束，保存在消息队列中的数据并没有消失，也就是说下次再从这个消息队列读数据的时候，就是上次的数据！！！
- 信号量， 不能传递复杂消息，只能用来同步
- 共享内存，只要首先创建一个共享内存区，其它进程按照一定的步骤就能访问到这个共享内存区中的数据，当然可读可写；

几种方式的比较：

- 管道：速度慢，容量有限
- 消息队列：容量受到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题。
- 信号量：不能传递复杂消息，只能用来同步
- 共享内存区：能够很容易控制容量，速度快，但要保持同步，比如一个进程在写的时候，

另一个进程要注意读写的问题，相当于线程中的线程安全，当然，共享内存区同样可以用作线程间通讯，不过没这个必要，线程间本来就已经共享了一块内存的。

第三题：
线程同步指多个线程同时访问某资源时，采用一系列的机制以保证同时最多只能一个线程访问该资源。线程同步是多线程中必须考虑和解决的问题，因为很可能发生多个线程同时访问（主要是写操作）同一资源，如果不进行线程同步，很可能会引起数据混乱，造成线程死锁等问题；
线程同步的方式：

- 临界区：通过对多线程的串行化来访问公共资源或者一段代码，速度快，适合控制数据访问
- 互斥量：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限，因为互斥对象只有一个，所以可以保证公共资源不会同时被多个线程访问
- 信号量：它允许多个线程同一时刻访问同一资源，但是需要限制同一时刻访问此资源的最大线程数目。信号量对象对线程的同步方式与前面几种方法不同，信号允许多个线程同时使用共享资源，这与操作系统中 PV 操作相似。
- 事件（信号）：通过通知操作的方式来保持多线程的同步，还可以方便的实现多线程的优先级比较的操作

总结比较：

- 互斥量与临界区的作用非常相似，但互斥量是可以命名的，也就是说它可以跨越进程使用。所以创建互斥量需要的资源更多，所以如果只为了在进程内部是用的话使用临界区会带来速度上的优势并能够减少资源占用量。因为互斥量是跨进程的互斥量一旦被创建，就可以通过名字打开它。
- 互斥量（Mutex），信号灯（Semaphore），事件（Event）都可以被跨越进程使用来进行同步数据操作，而其他的对象与数据同步操作无关，但对于进程和线程来讲，如果进程和线程在运行状态则为无信号状态，在退出后为有信号状态。所以可以使用 WaitForSingleObject 来等待进程和线程退出。
- 通过互斥量可以指定资源被独占的方式使用，但如果有下面一种情况通过互斥量就无法处理，比如现在一位用户购买了一份三个并发访问许可的数据库系统，可以根据用户购买的访问许可数量来决定有多少个线程/进程能同时进行数据库操作，这时候如果利用互斥量就没有办法完成这个要求，信号灯对象可以说是一种资源计数器。

第四题：
Threadlocal 和其他所有的同步机制都是为了解决多线程中的对同一变量的访问冲突，在普通的同步机制中，是通过对对象加锁来实现多个线程对同一变量的安全访问的。这时该变量是多个线程共享的，使用这种同步机制需要很细致的分析在什么时候对变量进行读写，什么时候需要锁定某个对象，什么时候释放该对象的索等等。所有这些都是因为多个线程共享了该资源造成的。Threadlocal 就从另一个角度来解决多线程的并发访问，Threadlocal 会为每一个线程维护一个和该线程绑定的变量副本，从而隔离了多个线程的数据共享，每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal 提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进 ThreadLocal。
总结：当然 ThreadLocal 并不能替代同步机制，两者面向的问题领域不同。同步机制是为了同步多个线程对相同资源的并发访问，是为了多个线程之间进行通信的有效方式；而 ThreadLocal 是隔离多个线程的数据共享，从根本上就不在多个线程之间共享资源（变量），这样当然不需要对多个线程进行同步了。所以，如果你需要进行多个线程之间进行通信，则使用同步机制；如果需要隔离多个线程之间的共享冲突，可以使用 ThreadLocal，这将极大地简化你的程序，使程

序更加易读、简洁。
第五题：
首先回答死锁的定义，所谓死锁就是一个进程集合中的多个进程因为竞争资源，而造成的互相等待现象。
死锁的原因：系统资源不足；多个进程的推进顺序不合理
死锁的必要条件：

- 互斥条件（Mutual exclusion）：资源不能被共享，只能由一个进程使用。
- 请求与保持条件（Hold and wait）：已经得到资源的进程可以再次申请新的资源。
- 非剥夺条件（No pre-emption）：已经分配的资源不能从相应的进程中被强制地剥夺。
- 循环等待条件（Circular wait）：系统中若干进程组成环路，改环路中每个进程都在等待相邻进程正占用的资源。

处理死锁的策略：

- 忽略该问题。例如鸵鸟算法，该算法可以应用在极少发生死锁的情况下。传说中，鸵鸟看到危险就把头深埋地下，这是显然是一种很消极的策略。
- 检测死锁并且恢复。
- 通过对资源有序分配，以避免循环等待的"环路"发生。
- 通过破坏死锁的必要条件，来防止死锁的产生。