Q1. What is deadlock?

在一组进程中，每一个进程都在等待被进程组中其他进程占用的资源，从而整个进程组都无法获取所有资源去执行也无法释放已有资源的状态。死锁的说法也可以用于线程，但此时资源主要指的是锁。

Q2. What are the requirements of deadlock?

1. Mutual exclusion：同一时刻只能有一个线程用到这个资源
2. Hold and wait：线程在等待被其他线程占用的资源时，不释放已经获得的资源
3. No preemption： 资源只能在线程执行完相关功能后被资源释放，不能强制释放
4. Circular wait：存在一组线程，其中每一个线程都在等待被其他线程持有的资源（线程 1 等待线程 2，2 等待 3，以此类推）

以上 4 个条件都是形成死锁的必要不充分条件

Q3. What's different between deadlock prevention and deadlock avoidance?

Deadlock prevention：在线程运行时就发生，破坏掉至少一个第二题中提出的死锁的形成条件。优点是，执行时不需要对死锁进行检测，节省了检测成本，也不需要持有系统资源分配状况的信息；缺点是系统的效率会降低。

Deadlock avoidance：在线程申请资源时发生，通过资源分配图、银行家算法等途径判断按申请分配资源是否会导致死锁，如果会导致死锁则不进行分配，反之则进行分配。优点是不会限制系统的性能；缺点是需要提前获得线程的资源请求，在每次资源分配都需要花费时间检测，同时还需记录系统的剩余资源数、进程已有资源数以及进程实现功能所需要的资源总数等信息。通过 avoidance 防止死锁，第二题中的四个条件在不产生死锁的情况下允许发生。

Q4. How to prevent deadlock? Give at least two examples.

至少破坏一个死锁形成所需要的条件。

例子：

1. 不允许进程在锁住已获得的资源的同时等待当前被其他进程占有的资源，比如当有部分资源被其他进程使用时就释放已有资源，一段时间后再尝试来请求资源。
2. 在进程在一开始就请求获得所有需要的资源，此时进程间不会持有共享的资源

Q5. Which way does recent UNIX OS choose to deal with deadlock problem, why?

忽视问题，假装系统中永远不会产生死锁。

原因：死锁发生的概率比较低，而处理死锁的花销比较大，因此不对死锁进行处理，将其抛给应用层（如数据库）或者用户解决。

Q6. What data structures you use in your implementation（of Banker's algorithm）? Where and why you use them? Are they optimal for your purpose?

使用了 struct 来表示进程以及进程信息，包括进程 PID 以及占有的资源。

使用 vector 来表示系统以及进程的资源状态，包括总余量以及每一个进程当前持有的资源数、最大需要的资源数等。

使用 map 将进程 PID 与程序中的进程计数器关联起来。

使用数组来存储其他进程信息，数组下标与进程计数器对应。

我认为这是最好的选择，每一个数据结构的功能都很明确，而且适用性很强，比如说使用

vector 可以实现资源种类数的动态声明。