姓名：胡瑞康

学号：22336087

ch18

1. 证明两阶段锁协议(Two-Phase Locking Protocol)能够保证冲突可串行化，且事务可以根据它们的锁点进行串行化。

步骤 1: 构建偏序关系

- 设有多个事务 T1, T2, ..., Tn，它们按照 2PL 协议执行。
- 假设事务 Ti 在其扩展阶段申请并获得锁 L，而事务 Tj 在其扩展阶段也需要锁 L，但 Ti 已经持有了锁 L。
  - 如果 L 是排他锁（X 锁），Tj 必须等待 Ti 释放 L。
  - 如果 L 是共享锁（S 锁），且 Ti 持有 X 锁，则 Tj 仍然需要等待。
- 因此，Ti 必须在 Tj 之前释放锁 L，才能让 Tj 继续执行。由此可得 $Ti \prec Tj$。

步骤 2: 偏序关系无环

- 由于 2PL 协议的扩展阶段和收缩阶段严格分离，并且锁只能在一个阶段申请和在另一个阶段释放，因此不会出现循环等待，避免了死锁。
- 偏序关系中无环路，保证可以进行拓扑排序。

步骤 3: 拓扑排序与串行化顺序

- 对偏序关系进行拓扑排序，得到一个线性顺序 S: T1, T2, ..., Tn。
- 这个顺序 S 表示事务的串行化执行顺序。

步骤 4: 并发执行与串行执行等价

- 证明并发执行的结果与按照 S 串行执行的结果相同。
- 由于 2PL 协议确保事务在获得所有需要的锁后才进入收缩阶段，且事务之间按照加锁顺序串行化，因此并发执行中事务的操作顺序与串行化顺序 S 中的操作顺序一致。
- 这意味着并发执行的结果与串行化顺序 S 的结果相同。

2. 在多粒度锁定中，隐式锁定和显式锁定有什么区别?

隐式锁定（Implicit Locking）

1. **自动获取**：隐式锁定是在执行某些操作时，系统自动获取的锁。例如，在数据库管理系统（DBMS）中，当执行一个查询或更新操作时，系统可能会自动对涉及的数据行或数据页加锁，以确保数据的一致性和完整性。
2. **粒度**：隐式锁定的粒度通常是较低的，比如行级锁定，系统根据操作的范围自动确定锁定的粒度。
3. **透明性**：对应用程序开发者来说，隐式锁定是透明的，不需要手动管理锁的获取和释放。
4. **优点**：简化了开发，减少了锁管理的复杂性，减少了死锁的可能性，因为锁定的粒度较小。
5. **缺点**：可能不够灵活，无法根据具体需求精确控制锁定的范围和粒度。

显式锁定（Explicit Locking）

1. **手动获取**：显式锁定是由程序员或用户显式地请求锁，通常通过特定的语句或 API 来实现。例如，在数据库中，可以使用 `SELECT ... FOR UPDATE` 语句显式地对某些行加锁。
2. **粒度**：显式锁定的粒度可以更灵活，可以根据需要锁定更粗粒度（如表锁）或更细粒度（如行锁）的对象。
3. **控制性**：显式锁定提供了更大的控制权，开发者可以精确地控制锁定的范围、时间和方式。
4. **优点**：提供了更大的灵活性和控制能力，可以根据具体应用的需求来优化并发性能。
5. **缺点**：增加了开发复杂性，需要开发者手动管理锁的获取和释放，容易出错，可能导致死锁或其他并发问题。

对比:

- **隐式锁定**是由系统自动管理的，粒度较小，透明性高，简化了开发但灵活性较低。
- **显式锁定**需要开发者手动管理，粒度更灵活，提供了更大的控制权，但增加了开发的复杂性和潜在的错误风险。

  。

3.

> 考虑以下两个事务：
> T34:
> read(A);
> read(B);
> if A = 0 then B := B + 1;
> write(B).
> T35:
> read(B);
> read(A);
> if B = 0 then A := A + 1;
> write(A).
> 为两个事务添加锁定（lock）和解锁（unlock）的指令，使事务 T34 和 T35 遵循两阶段锁协议(Two-Phase Locking Protocol)。说明这样会不会导致死锁。

T34:

```
lock A;
lock B;
read(A);
read(B);
if A = 0 then
    B := B + 1;
write(B);
unlock B;
unlock A;
```

T35:

```
lock A;
lock B;
read(B);
read(A);
if B = 0 then
    A := A + 1;
write(A);
unlock B;
unlock A;
```

不会导致死锁。因为两个事务都按照相同的顺序（先锁 A，再锁 B）获取锁，因此不会出现循环等待的情况。例如，如果 T34 先获取锁 A 和 B，T35 在等待获取锁 A，而 T34 释放锁后，T35 可以继续获取锁并执行。因此，这种加锁顺序是安全的，不会产生死锁。