Lab5 S2PL

在 lab 5 中，需要你基于现有的框架实现严格的两阶段锁协议：每个事务进行读的时候申请 Shared lock, 在写的时候申请 Exclusive lock。假如拿到锁才可以对数据进行读写，写的类型支持UPDATE和DELETE，事务会先将写的修改维护在本地的 write set 中，并且在 Commit 的阶段写到文件中。严格两阶段锁要求事务只有在 Commit/Abort 阶段才可以释放锁。

代码框架注释

Txn: 维护两个锁集合(shared/exclusive)，维护该事务的改动(write set)，读操作的时候会先遍历write set并且apply change

LockTable:

- txn_item：事务和申请的锁，有两种状态：granted / wait，假如事务在wait则该线程会阻塞等待
- txn_list：txn_item的链表，用于判断某个txn_item是否应该被授予锁
- LockTable: 严格两阶段锁的主要数据结构是 Lock Table, 其中会维护数据 Entry 到 txn_list的映射，主要的方法是request_lock & check_grant

TxManager: 事务管理器，理论上管理LockTable与Logging（但后者我们没有）管理事务的开始，每次id递增，Commit和Abort

TODO 10Pt

函数 fill in， 具体要求和hint在代码注释中给出

1. LockTable::request_lock 每个事务会根据读写类型调用request_lock函数
2. LockTable::Unlock Commit/Abort阶段需要调用Unlock，清理LockTable中该事务的所有锁
3. txn_list::check_grant (LockTable.h) 当一个事务加到LockTable后检查是否可以被授予锁

测试注释

多线程测试，每个线程会管理一个事务。首先解释一下，多线程的执行顺序是乱序的，但为了测试结果统一，我们给出了一个明确的执行顺序，如下图所示。测试中两个线程会严格按照给定的schedule执行，这一点通过进程间交换一些信号量实现。每次读取中会验证结果是否正确

Txn #0	**Txn #1**
BEGIN	
W(X)	
R(X)	
	BEGIN
	R(X)
COMMIT	
	R(X)
	COMMIT

为了避免阻塞等待（即没有拿到锁导致线程wait，无法向后执行），每个事务的主线程会开一些子线程进行R(X) W(X)的执行（这时没拿到锁的子线程将会等待，而程序也会继续执行，后续释放锁和加锁）因此你会看到在test_task_x的函数中有分子线程执行，最后调用join()进行merge操作的部分。

一共有两个测试，其中有必要的注释，推荐使用gdb，可以进行多线程debug

Good Luck!