Lab2 开发文档

19302010074 张诗涵

一、Lock

1) 实现思路：主要使用了面包店算法（我认为可以算是扩展到多线程情况下的 peterson 算法）

用一个 level 数组来保存线程目前的优先级（0 是默认情况，即对锁没有兴趣）

用一个 choosing 数组来标志当前线程是否正在等待分配优先级。

由于不允许给 lock 和 unlock 函数传入参数，因此记录到 level 数组时需要通过 Thread.currentThread.getId()得到当前线程 id，并通过 mod level.length 的方式映射到对应的 level 大小的空间上。为了减少撞 key 的可能，这里将 level 的大小初始化为了线程数的 10 倍。

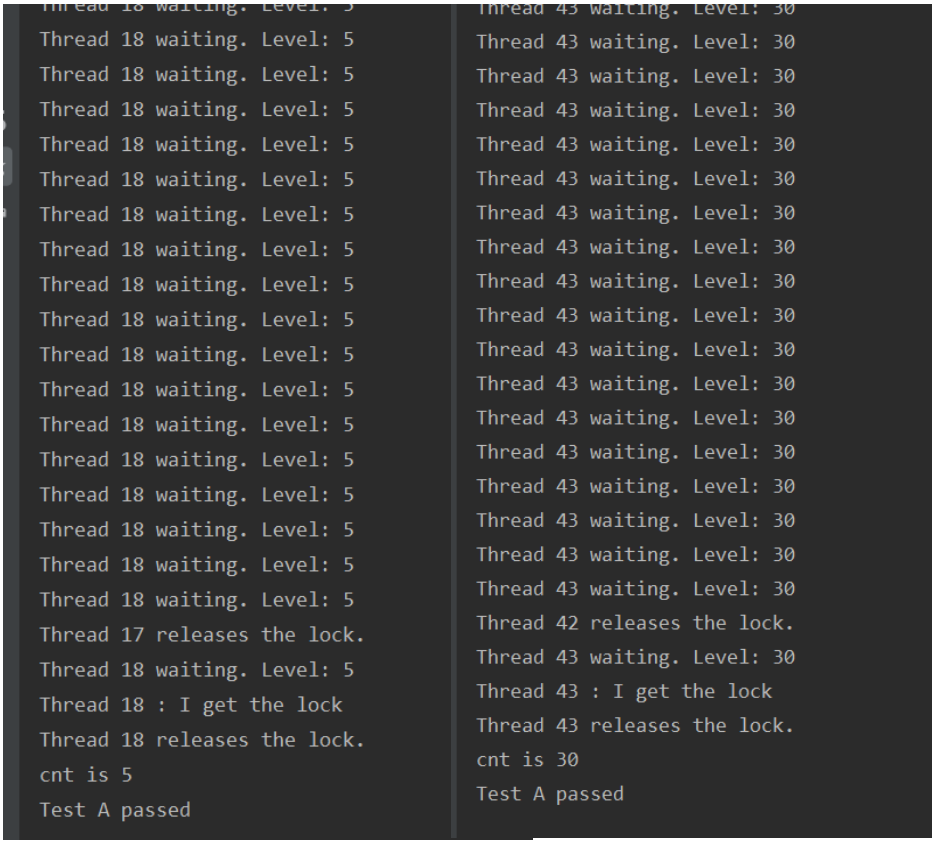
获取 level：首先将 choosing 中 id 对应位置设为 true，示意其他试图获取锁的兄弟进程不要抢。接着得到 level 数组中目前的最大值，并将其+1 作为此进程的 level

尝试获取 lock：遍历全部进程，如果目前进程尚在等待分配优先级，就等待其分配好，接着比较自己与它的优先级、如果更高就跳出对它的等待进入对下一个比较；总体而言，在该线程的优先级成为全部感兴趣线程中最高的时候它就能跳出循环得到锁。

释放 lock：将自己位置的 level 恢复为默认值 0。

2) 运行结果：

在五个线程的情况下可以正确完成（下图左）：



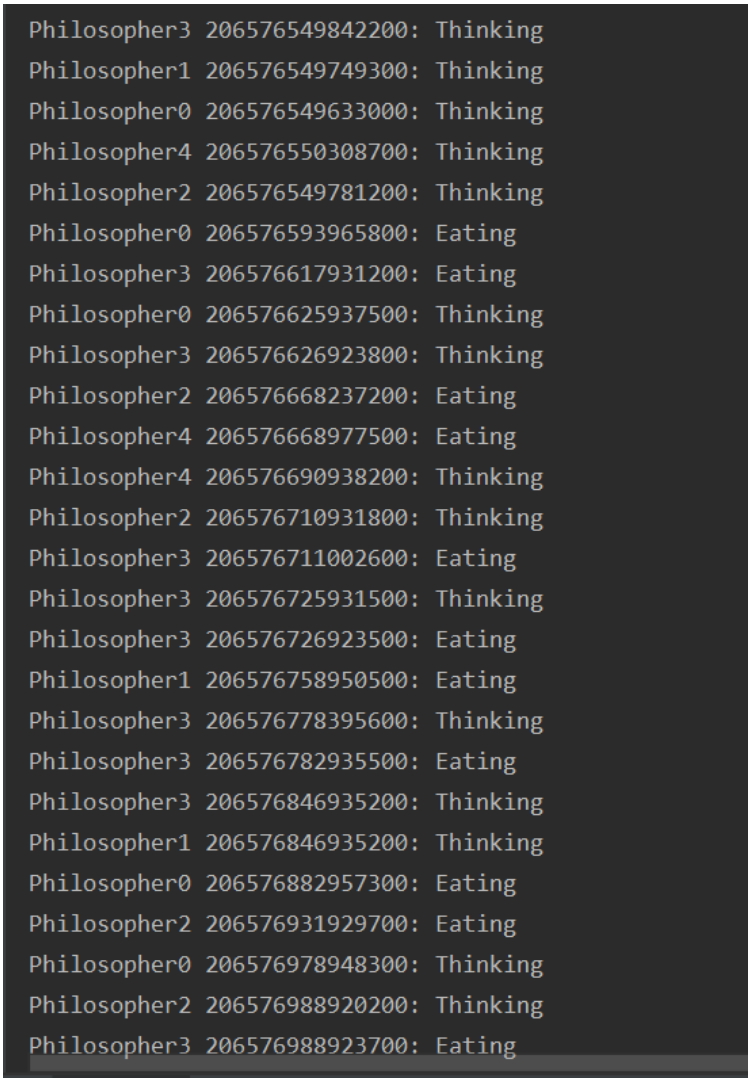
同样的，在扩展到更多的线程下也可以正确完成（上图右）

二、哲学家吃饭问题

1) 实现思路：

如果按每个人都先拿起左手边叉子再拿起右手边叉子操作，会出现死锁的情况就是当五个人同时拿起了各自左手的叉子，此时每个人都在等待右手边的叉子、但是没有人会主动放下手中的叉子给自己左边的那个人，此时就出现了死锁。那么解决的方法也很简单，破坏死锁的循环等待条件即可。我们不再按左手边右手边顺序拿起筷子，而是通过给筷子添加 id 实现固定的全局顺序获取，根据 id 从小到大获取，不会出现死锁情况。

2) 运行结果：



可以看出哲学家之间可以互相交替地完成吃饭行为而不会出现死锁、饥饿等问题。