HW10

522031910213 朱涵

May 16, 2024

1 PART1

- 死锁：在多个进程间由于资源被占用而阻塞导致的无限等待的情况。拿两个进程举最简单的例子——进程A拿到了锁B，进程B拿到了锁A，但是进程A需要等到锁A被释放才能释放锁B，同理进程B需要等到锁B被释放才能释放锁A，那么两个进程就会永远阻塞；

- 活锁：在多个进程间由于竞争而不断重复操作的状态。还是拿两个进程举例–进程A和进程B都需要锁X，它们同时去拿X，但是同时发现对方也想拿X，于是两个进程很谦让，都选择了让对方拿锁，结果两个进程都没拿，然后继续检测到锁X可以获取，两个进程又同时去拿.....由此陷入了循环状态，无法推进有效的工作。与死锁不同的是，处于活锁的进程不是阻塞的，只是因为反复重复某些竞争操作而导致程序整体无法进行下去。另外，死锁在实际编程比较常见，并且可以通过确定资源释放顺序来解决。但是活锁是比较少见的，因为需要小概率的特殊竞争条件（比如两个进程一直同时拿取，同时释放），并且调度算法会有意的避免这种情况。

2 PART2

1. 最简单的方式就是采用互斥锁实现读者-写者模式。首先要使用mutex把**push**、**pop**操作进行封装，变成原子操作。也就是在进行对共享变量的操作（在这里主要是头节点或头指针的调用）前后用mutex锁保护起来。要注意，如果函数的参数也是共享变量（比如头节点本身），这样的访问也需要考虑在互斥锁之内。另外，如果有条件判断（比如头节点插入删除的特判），每个return语句之前都需要记得释放锁，否则程序会阻塞。至于**isEmpty**方法，因为是一个只读操作，所以可以和其他只读方法任意并发，但是由于写操作需要独占访问，因此考虑加上读者锁/写者锁来实现对于读操作/写操作的访问，即等待所有读者并发读完后再释放写者锁，或是等待一连串的写操作完后再释放读者锁。

2. 可能的性能瓶颈主要在于锁的竞争。锁的获取和释放是高开销的，如果写操作（push/pop）过多，会导致程序频繁地竞争锁资源，从而导致性能下降，甚至可能不如单线程的性能。另外，读者和写者的竞争也是需要考虑的，如果是多读场景，却设计了写者优先的模式，有可能导致写者的饥饿。

3. 由于存在多个锁，有可能会有死锁的情况，这时就需要设计正确的获取和释放顺序来解决问题。比如读者总是先获取写者锁，然后再获取操作的锁，那么释放时需要反过来释放，先获得的先释放。另外，就算只有一把锁也可能出现死锁，比如push操作中可能在容器满时调用扩容操作，而扩容操作也需要获得锁，此时调用前就需要释放锁确保不会阻塞。至于活锁，可能会在多个push/pop同时调用时发生，但是概率比较小，也可以通过超时机制或者最大重复尝试次数等来解决。