并行程序设计与算法 第三次作业

刘森元, 21307289

中山大学计算机学院

1 简答题

1.1 习题 1

如果一个程序使用超过一个互斥量, 并能够以不同的顺序来获取互斥量, 程序可能会死锁.

(1) 用两个线程运行程序, 假设发生了下列顺序的时间, 会发生什么?

时间	线程 0	线程 1
0	`pthread_mutex_lock(&mut0)`	`pthread_mutex_lock(&mut1)`
1	`pthread_mutex_lock(&mut1)`	`pthread_mutex_lock(&mut0)`

Table 1: 事件表

(2) 如果程序使用忙等待 (采用两个标志变量) 替代互斥量, 还会有问题吗?

(3) 如果程序使用信号量替代互斥量, 还会有问题吗?

(1) 在这种情况下, 可能会发生死锁. 线程 0 锁定了 `mut0`, 然后试图锁定 `mut1`. 同时, 线程 1 锁定了 `mut1`, 然后试图锁定 `mut0`. 如果两个线程都在等待对方释放其已锁定的互斥量, 那么就会发生死锁.

(2) 使用忙等待和两个标志变量替代互斥量可能仍然会有问题. 如果两个线程同时检查对方的标志并发现对方没有锁定资源, 然后两者都认为自己可以进入临界区, 这就可能导致竞态条件. 这种情况下, 两个线程可能同时进入临界区, 导致数据不一致.

(3) 使用信号量替代互斥量可能仍然会有问题. 信号量和互斥量的主要区别在于, 信号量可以有多个可用资源, 而互斥量只允许一个资源被锁定. 然而, 如果两个线程以不同的顺序请求信号量, 仍然可能发生死锁. 例如, 如果线程 0 获取了信号量 A, 然后试图获取信号量 B, 同时线程 1 获取了信号量 B, 然后试图获取信号量 A, 那么就可能发生死锁.

1.2 习题 2

考虑一个链表以及对链表进行的操作, 下列的哪些操作可能会导致问题, 如果会导致问题, 请举例说明:

(1) 两个 Delete 操作同时进行

(2) 一个 Insert 和一个 Delete 操作同时进行

(3) 一个 Member (查询一个节点是否存在) 和一个 Delete 操作同时进行

(4) 两个 Insert 同时进行

(5) 一个 Insert 和一个 Member 同时进行

(1) 会导致问题. 例如, 如果两个线程同时尝试删除同一个节点, 第一个线程可能会成功删除节点, 但第二个线程可能会尝试删除一个已经不存在的节点, 这可能导致未定义的行为.

(2) 会导致问题. 例如, 如果一个线程正在插入一个节点, 而另一个线程正在删除同一个位置的节点, 可能会导致链表状态的不一致.

(3) 会导致问题. 例如, 如果一个线程正在查询一个节点, 而另一个线程正在删除该节点, 查询操作可能会返回一个已经不存在的节点.

(4) 会导致问题. 例如, 如果两个线程同时尝试在同一个位置插入节点, 可能会导致链表状态的不一致.

(5) 不会导致问题, 只要插入操作在查询操作之前完成. 但是, 如果查询操作在插入操作完成之前运行, 查询操作可能无法找到新插入的节点, 即使它已经在链表中.

1.3 习题 3

> 链表操作 Insert 和 Delete 可以拆成两个阶段, 第一阶段两个操作都先找到要操作节点的位置, 在第二阶段才插入或删除一个节点; 也即第一阶段都只涉及对链表的读访问, 只有第二阶段才写访问链表. 如果在第一阶段使用一个读锁来锁链表, 在第二阶段使用一个写锁来锁链表, 假设该读写锁的实现是 「获取写锁需要先释放读锁」, 这样是否安全?

不安全. 虽然在第一阶段使用读锁可以防止多个线程同时修改链表, 但在释放读锁和获取写锁的间隙, 其他线程可能会修改链表, 这可能导致第二阶段的操作基于过时或无效的信息.

例如, 考虑两个线程, 一个执行插入操作, 另一个执行删除操作. 两者都在第一阶段找到了要操作的节点位置, 然后释放了读锁. 然后, 删除操作获取写锁并删除节点, 然后释放写锁. 此时, 插入操作获取写锁并尝试在已经被删除的节点位置插入新节点, 这可能导致未定义的行为.

1.4 习题 4

> 在矩阵-向量乘法的例子中, 采用 8 000*8 000 的输入, 假设程序用 4 个线程运行, 线程 0 和线程 2 被分配到不同的处理器上运行. 如果一个缓存行大小为 64 字节或 8 个 double 数, 在线程 0 和线程 2 之间会对向量 y 的任何一部分发生伪共享吗? 如果线程 0 和线程 3 被分配到不同的处理器, 会发生伪共享吗?

假设矩阵-向量乘法的并行实现是这样的：每个线程处理矩阵的一部分行, 并将结果累加到对应的向量 y 的元素中.

1. 对于线程 0 和线程 2：由于矩阵被分割为 4 个部分, 线程 0 处理前 2000 行, 线程 2 处理第 4001 到 6000 行. 因此, 线程 0 和线程 2 更新向量 y 的不同部分, 它们不会对同一个缓存行进行写操作. 所以, 线程 0 和线程 2 之间不会发生伪共享.

2. 对于线程 0 和线程 3：线程 0 处理前 2000 行, 线程 3 处理第 6001 到 8000 行. 同样, 它们更新向量 y 的不同部分, 不会对同一个缓存行进行写操作. 所以, 线程 0 和线程 3 之间也不会发生伪共享.

1.5 习题 5

> 在矩阵-向量乘法的例子中, 采用 8*8 000 000 的输入, 假设一个缓存行的大小与习题 4 相同, 同时假设系统有 2 个双核处理器, 假设同一个处理器上的所有和共享一个缓存.

(1) 如果只考虑一对线程共享一个处理器, 可以有多少种不同的方式将 4 个线程分配到处理器上?

(2) 在第 (1) 问的线程分配方式中, 是否有一种线程分配方式以及一种向量元素到缓存行的分配方式, 使得没有伪共享的发生?

y[0]	**y[1]**	**y[2]**	**y[3]**	**y[4]**	**y[5]**	**y[6]**	**y[7]**

Table 2: 分配方式 1

?	**?**	**?**	**?**	**?**	**?**	**?**	**y[0]**
y[1]	y[2]	y[3]	y[4]	y[5]	y[6]	y[7]	?

Table 3: 分配方式 2

(1) 如果只考虑一对线程共享一个处理器, 那么有 $C_4^2 = 6$ 种选择方式.

(2) 在第 (1) 问的线程分配方式中, 是有一种线程分配方式以及一种向量元素到缓存行的分配方式, 使得没有伪共享的发生. 具体来说, 我们可以将线程 0 和线程 2 分配到一个处理器上, 线程 1 和线程 3 分配到另一个处理器上. 然后, 我们可以按照 "分配方式 1" 将向量元素分配到缓存行上. 这样, 同一个处理器上的线程会更新不同的缓存行, 因此不会发生伪共享.

对于 "分配方式 2" , 如果我们将线程 0 和线程 1 分配到一个处理器上, 线程 2 和线程 3 分配到另一个处理器上, 那么也不会发生伪共享. 因为在这种情况下, 同一个处理器上的线程会更新不同的缓存行.