第六章课后练习

6.8

假设有两个买家同时分别出价 amount1 和 amount2（amount1>amount2>highestBid）并调用bid方法，且其指令执行顺序如下时，会出现race condition：

① if (amount > highestBid)　//amount=amount1，if内条件为真

② if (amount > highestBid)　//amount=amount2，if内条件为真

③ highestBid = amount;　　//highestBid=amount1

④ highestBid = amount;　　//highestBid=amount2

最终highestBid的值为amount2，但正确的执行结果应该是highestBid=amount1。

可能的避免方式：通过设置信号量的方式，将bid函数调用放于临界区内，使得每次只有一个进程可以调用bid函数。

6.13

① mutual exclusion：当 Pi 即将进入临界区时，flag[j]==false，此时 Pj 必定处于exit区，刚执行完turn=i，flag[j]=false这两个语句（不会有其他情况可以改变turn和flag[j]的值），由于此时flag[i] ==true，因此Pj后续会进入内嵌第一层的while循环，并进行忙等待。由于在Pi执行临界区代码时，flag[i] ==true和turn ==i 的条件始终成立，因此Pj始终会在while循环中进行忙等待，无法进入临界区。同理，Pj进入临界区的时候，Pi也会等待而无法进入临界区，保证了互斥。

② progress：当Pi执行完临界区时，在exit区中将执行turn=j，flag[i]=false这两个语句，一经执行则Pj中 turn == i 和flag[i]的条件就不成立了，Pj会执行flag[j] = true让Pi陷入忙等待，并跳出while循环开始执行临界区。同理当Pj执行完临界区时也会由于类似的原因陷入忙等待，且令Pi开始执行临界区。因此该方法可以保证进程不会永远得不到执行的机会，满足progress的要求。

③ bounded waiting：由②得，当Pi在等待Pj时，只要Pj执行完临界区内容并进入exit区，Pi就可以跳出while循环并开始执行临界区。由于我们假设Pj的临界区内容总可以在有限时间内执行完，因此Pi的等待时间也是有限长的。同理Pj的等待时间也是有限长的。符合bounded waiting的要求。

6.21

第二种，使用原子变量的方式更加高效，理由如下：

在第一种使用mutex的方式中，n个需要改变hits值的线程中，有n-1个都会在请求可用锁的过程中陷入忙等待，这是对CPU时间的极大浪费，尤其是在线程数很大的情况下更是如此。而在第二种使用原子变量的方式中，多个线程之间在有序的进行对hits值的操作，不会有额外的CPU时间被浪费，因此效率更高。