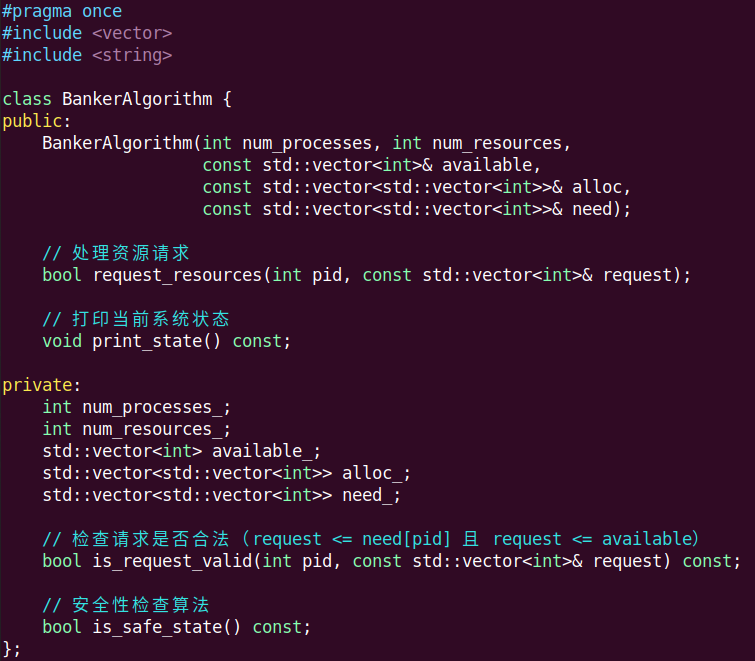
操作系统第三次上机

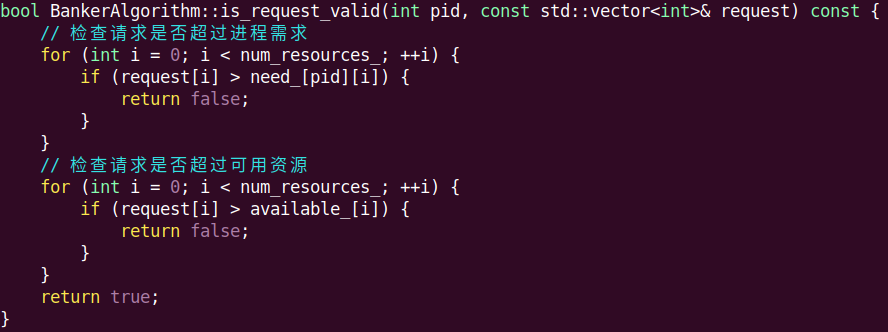
本次上机的任务是解决银行家算法的问题，银行家问题在于避免死锁的情况发生，多用于多进程共享有限资源的系统中，举例的模型是现在又n个进程，他们需要争夺m类有限的资源，我们用类定义银行家算法，private成员是进程的数量num\_processes\_，资源的种类数是num\_resources\_，我们用三个数组来分别存放每种资源的剩余个数，已分配的资源，以及所需要的资源分别用vector<int> available\_;vector<vector<int>> alloc\_;vector<vector<int>> need\_;来表示，至于为什么后两个数组用的是二维数组，因为已分配的资源不仅要表示分配的进程PID，还要表示分配的多少，需求同理，不仅要表示需求进程的PID还要表示需求量的多少。第一种用下标表示PID即可

类和头文件的定义如下所示：Banker.hpp

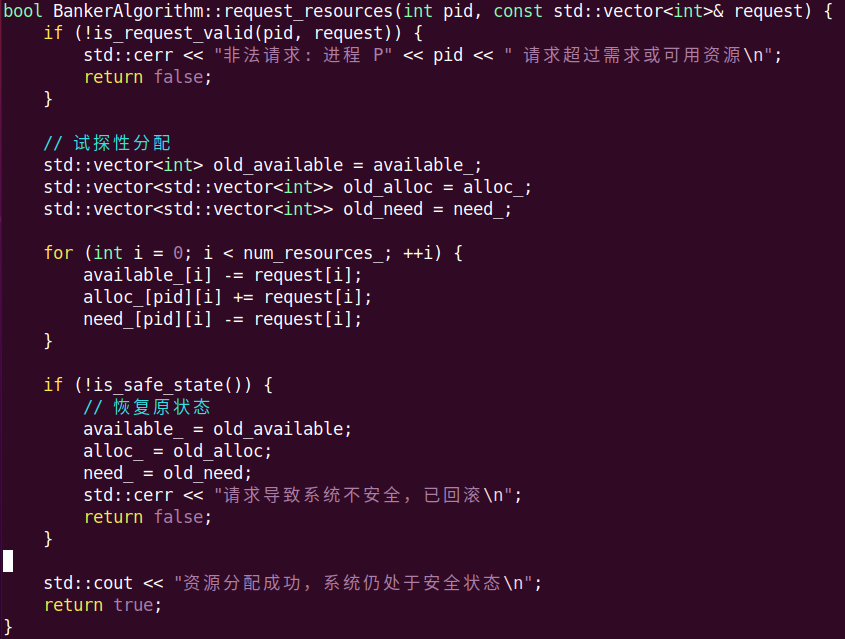


这些函数的功能分别是用来处理资源请求，表示系统当前状态以及检查请求合法性，安全性检查。

实现的逻辑和代码如下：



这是判断请求是否合法的函数：首先遍历资源种类，看看对每种请求是不是超过了需求的限制，再检查请求是不是超过了可用的资源数，如果超过了就返回false。

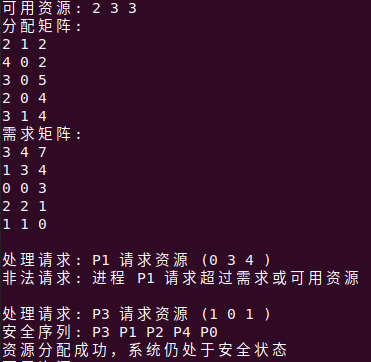


这是申请资源的函数，来做出安全的分配策略。基于银行家算法，我们知道需要保证整个程序的安全性，也就是说得先判断程序需求的资源有没有超过银行资源的最大剩余量。如果超过了无法“拨款”,如果程序“借钱”的数值在银行的承担范围之内，就先试着给这个程序分配资源。然后用一个函数来判断这样分配之后是否安全—即不会造成死锁的现象。如果不安全就回退，安全就分配成功。



这是判断是否安全的函数，首先用一个数组保存available的数据，然后把能通过的程序安排成true用一个bool类型的finish数组存储每个进程的标记，先默认初始化为false。

接下来用一个三层循环来遍历所有的进程，至于为什么是三层循环，我们先看内两层循环的逻辑，第一层遍历所有的进程，进入第二层的条件是finish是false也就是还没确定是否安全的进程。第二层遍历每个进程所需要的资源种类，如果可以释放的话就把目前的内存释放给work[j]，把finish改成true代表这个程序安全了，把found改为true，然后直接跳出循环。第三层的循环代表的是找到安全的序列，由于最终的目的是让finish的元素全部变成1，而释放过的程序的资源可以给后序的程序使用，就需要用第三层循环把每一次都遍历一遍。最后设置的if(!found) break;这个判断是保证如果有一次进入内层循环没找到安全的进程就直接break掉跳出循环，说明这个资源分配是不安全的。

最后主函数来测试一个案例，并给出数据：

