操作系统实验四实验报告
银行家算法

张子扬 无06 2020010790

一、 问题描述

银行家算法是操作系统的经典算法之一，用于避免死锁情况的出现。它最初是为银行设计的（因此得名），通过判断借贷是否安全，然后决定借不借。在操作系统中，银行家、出借资金、客户，就分别对应操作系统、资源、申请资源的进程。

二、 实验平台

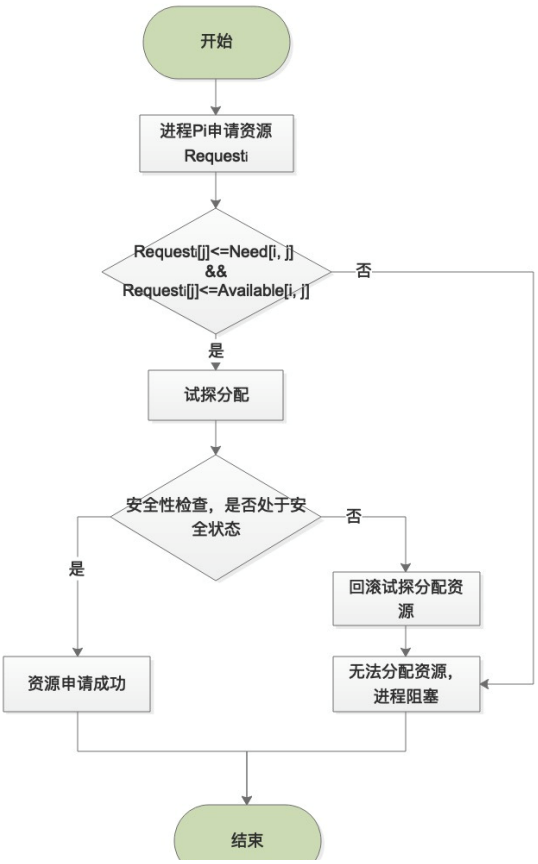
本实验在 Python3.10.0 上运行。

三、 实验原理

每一个新进程进入系统时，必须声明需要每种资源的最大数目，其数目不能超过系统所拥有的的资源总量。当进程请求一组资源时，系统必须首先确定是否有足够的资源分配给该进程，若有，再进一步计算在将这些资源分配给进程后，是否会使系统处于不安全状态。如果不会才将资源分配给它，否则让进程等待。

为了实现银行家算法，在系统中必须设置这样四个数据结构：1）Available 向量：系统中可利用的资源数目；2）Allocation 矩阵：每个进程已分配的各类资源的数目；3）Need 矩阵：每个进程还需要的各类资源数。

四、 实验流程图



五、 算法实现

实现安全性检查算法

```
def is_safe(self):
        work = self.available[:]
        finish = [False] * self.n_processes
        while True:
            found = False
            for i in range(self.n_processes):
                if not finish[i] and all(self.need[i][j] <=
work[j] for j in range(self.n_resources)):
                    for j in range(self.n_resources):
                        work[j] += self.allocation[i][j]
                    finish[i] = True
                    found = True
                    print("In safe check: process {} is
finished.".format(i))
            if not found:
                break
        return all(finish)
```

实现分配算法：

```
def request(self, process_id, request_vector):
        if any(request_vector[i] > self.need[process_id][i] for
i in range(self.n_resources)):
            return (False, "requesting more resources than
needed.")
        if any(request_vector[i] > self.available[i] for i in
range(self.n_resources)):
            return (False, "not enough resources.")
        for i in range(self.n_resources):
            self.available[i] -= request_vector[i]
            self.allocation[process_id][i] += request_vector[i]
            self.need[process_id][i] -= request_vector[i]
        safe = self.is_safe()
        if not safe:
            for i in range(self.n_resources):
                self.available[i] += request_vector[i]
                self.allocation[process_id][i] -=
request_vector[i]
                self.need[process_id][i] += request_vector[i]
        return (safe, "banker is safe." if safe else "banker is
unsafe.")
```

六、 运行测例

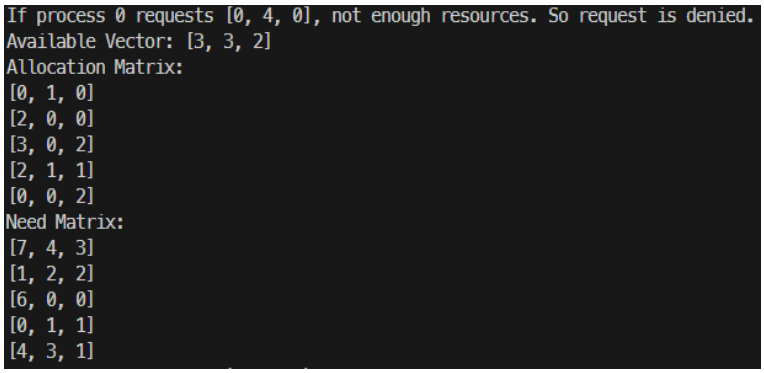
设计四组测试用例，分别对四种情况进行测试：当前资源不足而等待、因申请资源超出需求而报错、因会产生死锁而等待、成功分配。

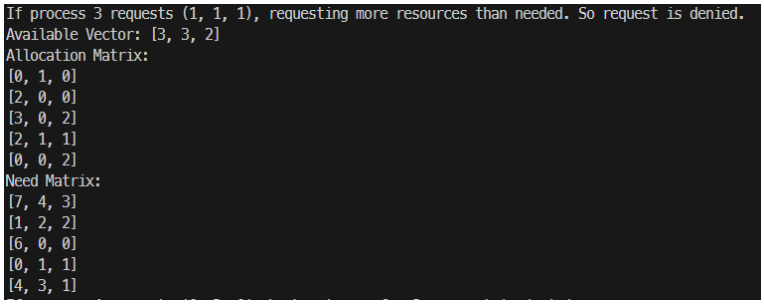
首先规定 need、allocation 矩阵，available 向量。

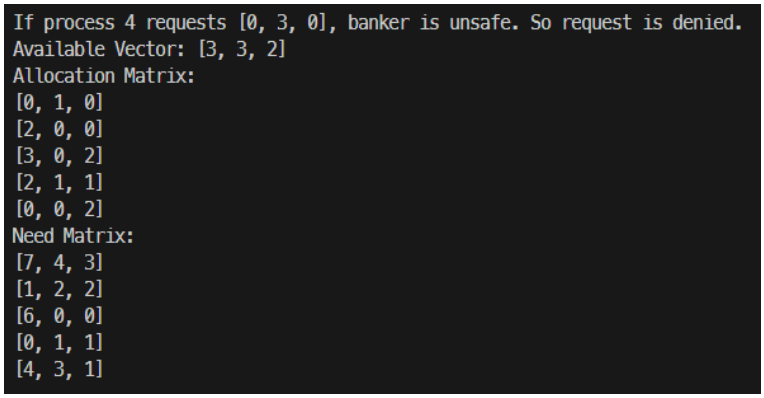
$$Need = \begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{bmatrix}, Allocation = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

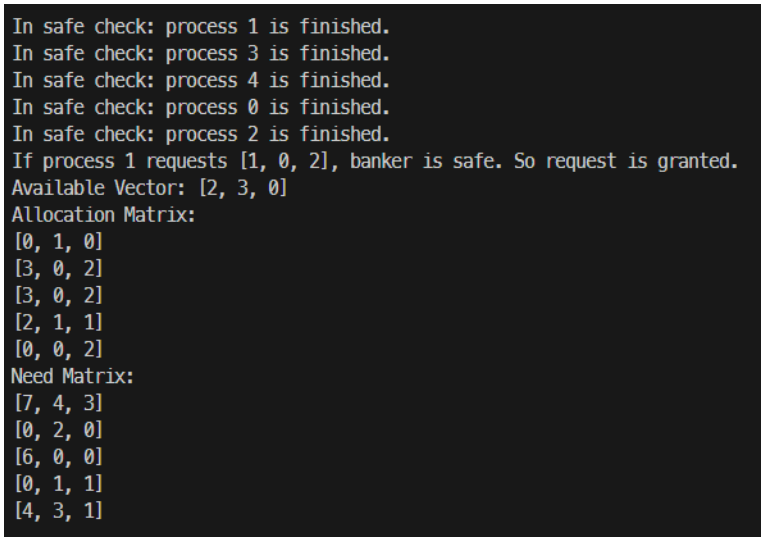
$$Available = [3 \quad 3 \quad 2]$$

之后运行测例，结果如图所示。









上面的结果与课堂上所讲的银行家算法分析结果一致，证明了实验中算法逻辑的正确性。

七、 算法鲁棒性和效率分析

本算法对边界情况可以进行处理。在 main 中演示了当进程数为 0 时，算法是可以得出安全的结论的，与理论知识相符。由此可见，该算法的鲁棒性较为不错。

在算法效率上，设资源数为 m、进程数为 n，执行安全性算法时，每次扫描所有进程中的所有资源，复杂度为 nm，而扫描要进行n次。因此算法复杂度为mn^2。

八、 思考题

银行家算法在实现过程中需注意资源分配的哪些事项才能避免死锁？

（1） 进程对资源的申请数量不能超过其所需数量。

（2） 进程对资源的申请数量不能超过系统当前可用的资源总量。

（3） 分配资源后，系统不能进入不安全状态。