实验 4 处理机调度

无 16 李根 2021012787

银行家算法

问题描述

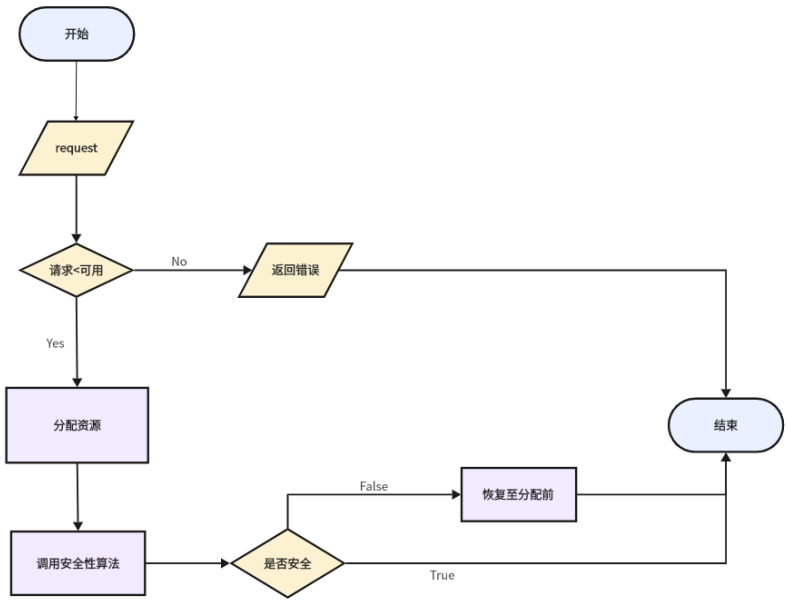
银行家算法是避免死锁的一种重要方法，将操作系统视为银行家，操作系统管理的资源视为银行家管理的资金，进程向操作系统请求分配资源即企业向银行申请贷款的过程。请根据银行家算法的思想，编写程序模拟实现动态资源分配，并能够有效避免死锁的发生。

实现要求

1．对实现的算法通过流程图进行说明；
2．设计不少于三组测试样例，需包括资源分配成功和失败的情况；
3．能够展示系统资源占用和分配情况的变化及安全性检测的过程；
4． 结合操作系统课程中银行家算法理论对实验结果进行分析，验证结果的正确性；
5．分析算法的鲁棒性及算法效率

具体实现

对于银行家算法，我们有较为明确得到流程，简要示意图如下所示：



为了编程实现代码，我先定义了一个 BankersAlgorithm 类，初始化时我们需要提供三个参数：可用资源向量 A（Available），各线程最大需求矩阵

Max_Demand 以及当前各个进程的分配矩阵 Allocation

此外，我们需要让类自动地计算好需求矩阵 Need：

```
def __init__(self, available, max_demand, allocation):
    self.available = available
    self.max_demand = max_demand
    self.allocation = allocation
    self.num_processes = len(max_demand)
    self.num_resources = len(available)
    self.need = self.Need()

1 usage
def Need(self):
    need = []
    for i in range(self.num_processes):
        need.append([self.max_demand[i][j] - self.allocation[i][j] for j in range(self.num_resources)])
    return need
```

事实上，对于银行家算法，在具备了上面的数据结构之后，我们所需要实现的其他功能也就只剩下了响应进程请求与判断是否存在不安全状态两个。

我们采用的判断不安全状态的函数是从贪心角度出发的，对于某一时刻的算法状态，我们对没有完成分配的进程进行遍历，当每一次请求结束时，我们调用安全状态判断的函数，如果此时剩下的状态不安全，则需要回滚至请求之前，并拒绝此次请求；安全则返回安全序列。

具体的实现函数如下所示，这些都是 BankersAlgorithm 类的方法。

```
def is_safe(self):
    work = self.available[:]
    finish = [False] * self.num_processes
    safe_sequence = []

    while len(safe_sequence) < self.num_processes:
        allocated = False
        for i in range(self.num_processes):
            if not finish[i]:
                if all(self.need[i][j] <= work[j] for j in range(self.num_resources)):
                    for k in range(self.num_resources):
                        work[k] += self.allocation[i][k]
                    finish[i] = True
                    safe_sequence.append(i)
                    allocated = True
        if not allocated:
            return False, []
    return True, safe_sequence
```

```
def request_resources(self, process_id, request):
    # 检查请求是否满足需求和可用资源
    if all(request[i] <= self.need[process_id][i] for i in range(self.num_resources)) and \
            all(request[i] <= self.available[i] for i in range(self.num_resources)):
        # 假设分配资源
        for i in range(self.num_resources):
            self.available[i] -= request[i]
            self.allocation[process_id][i] += request[i]
            self.need[process_id][i] -= request[i]

        safe, _ = self.is_safe()
        if not safe:
            for i in range(self.num_resources):
                self.available[i] += request[i]
                self.allocation[process_id][i] -= request[i]
                self.need[process_id][i] += request[i]
            return False
        return True
    else:
        return False
```

这些函数中均使用了 all()函数作为判断。实现起来还是比较容易的。

测试结果

我们先初始化一个算法类 ba，定义输入参量如下：

```
available = [3, 3, 2]
max_demand = [
    [7, 5, 3], [3, 2, 2], [9, 0, 2], [2, 2, 2], [4, 3, 3]
]
allocation = [
    [0, 1, 0], [2, 0, 0], [3, 0, 2], [2, 1, 1], [0, 0, 2]
]
ba = BankersAlgorithm(available, max_demand, allocation)
```

根据算法可能返回的三种情况，我设计了三种不同的输入：

（1）正确请求

```
Test Case 1: Process 1 requests [1, 0, 2]
Request result: True
Available: [2, 3, 0]
Allocation: [[0, 1, 0], [3, 0, 2], [3, 0, 2], [2, 1, 1], [0, 0, 2]]
Need: [[7, 4, 3], [0, 2, 0], [6, 0, 0], [0, 1, 1], [4, 3, 1]]
```

进程 1（allo:[2,0,0]）请求分配[1,0,2]，能够安全请求，系统也完成了分配，结果如上图。

（2）资源不足导致的失败

```
Test Case 2: Process 4 requests [3, 3, 0]
Request result: False
Available: [2, 3, 0]
Allocation: [[0, 1, 0], [3, 0, 2], [3, 0, 2], [2, 1, 1], [0, 0, 2]]
Need: [[7, 4, 3], [0, 2, 0], [6, 0, 0], [0, 1, 1], [4, 3, 1]]
```

可以看出此时的 Available 并不能满足进程 4 的[3,3,0]的请求，算法没有分配资源，结果如上图

（3）不安全状态导致的失败

```
Test Case 3: Process 0 requests [0, 2, 0]
Request result: False
Available: [2, 3, 0]
Allocation: [[0, 1, 0], [3, 0, 2], [3, 0, 2], [2, 1, 1], [0, 0, 2]]
Need: [[7, 4, 3], [0, 2, 0], [6, 0, 0], [0, 1, 1], [4, 3, 1]]
```

此时为 0 进程请求资源[0,2,0]，资源显然是足够的，但可以看出如果将资源分配给进程 0 之后，所有进程将都无法获得足够的资源完成任务，导致了死锁，因此系统没有给这一请求分配资源，所有资源回滚至了请求前。

算法鲁棒性与效率分析

综上所述，我的代码正确实现了系统占用与安全性检测，顺利实现了银行家算法。从多次测试来看，算法鲁棒性还是较强的，能够应对很多不安全的请求；但为了保证算法的鲁棒性，我选择使用回滚的方式来实现安全性检测，这一方法可能算不上有很高的效率。

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？

在实验过程中，我为了防止死锁做了如下操作：

（1）在分配资源前，使用银行家算法检查是否会导致系统进入不安全状态。如果会，拒绝资源请求以避免死锁。

（2）确保进程请求的资源数量是完整的，不能部分满足请求。

（3）当进程完成其任务后，保证所有分配给它的资源被算法正确释放，同时更新算法的相关矩阵量

（4）保证算法在初始状态也需要出于安全状态

总结

银行家算法我们可以根据课件上的伪代码进行改写，便可以实现。实现的关键也就是在算法中我们需要对不安全状态的判定进行分析。通过本次实验，我也对这一算法有了更深入的了解。感谢老师与助教的指导！

代码如下

```
class BankersAlgorithm:
    def __init__(self, available, max_demand, allocation):
        self.available = available
        self.max_demand = max_demand
        self.allocation = allocation
        self.num_processes = len(max_demand)
        self.num_resources = len(available)
        self.need = self.Need()

    def Need(self):
        need = []
        for i in range(self.num_processes):
            need.append([self.max_demand[i][j] -
self.allocation[i][j] for j in range(self.num_resources)])
        return need

    def is_safe(self):
        work = self.available[:]
        finish = [False] * self.num_processes
        safe_sequence = []

        while len(safe_sequence) < self.num_processes:
            allocated = False
            for i in range(self.num_processes):
                if not finish[i]:
                    if all(self.need[i][j] <= work[j] for j in
range(self.num_resources)):
                        for k in range(self.num_resources):
                            work[k] += self.allocation[i][k]
                        finish[i] = True
                        safe_sequence.append(i)
                        allocated = True
            if not allocated:
                return False, []
        return True, safe_sequence

    def request_resources(self, process_id, request):
        # 检查请求是否满足需求和可用资源
        if all(request[i] <= self.need[process_id][i] for i in
range(self.num_resources)) and \
                all(request[i] <= self.available[i] for i in
```

```
range(self.num_resources)):
            # 假设分配资源
            for i in range(self.num_resources):
                self.available[i] -= request[i]
                self.allocation[process_id][i] += request[i]
                self.need[process_id][i] -= request[i]

            safe, _ = self.is_safe()
            if not safe:
                for i in range(self.num_resources):
                    self.available[i] += request[i]
                    self.allocation[process_id][i] -= request[i]
                    self.need[process_id][i] += request[i]
                return False
            return True
        else:
            return False


available = [3, 3, 2]
max_demand = [
    [7, 5, 3], [3, 2, 2], [9, 0, 2], [2, 2, 2], [4, 3, 3]
]
allocation = [
    [0, 1, 0], [2, 0, 0], [3, 0, 2], [2, 1, 1], [0, 0, 2]
]
ba = BankersAlgorithm(available, max_demand, allocation)

# 测试样例
#成功请求样例
result = ba.request_resources(1, [1, 0, 2])
print("Request result:", result)
print("Available:", ba.available)
print("Allocation:", ba.allocation)
print("Need:", ba.need)

#请求资源失败（资源不足）
result = ba.request_resources(4, [3, 3, 0])
print("Request result:", result)
print("Available:", ba.available)
print("Allocation:", ba.allocation)
print("Need:", ba.need)

# 请求资源失败（不安全状态）
```

```
result = ba.request_resources(0, [0, 2, 0])
print("Request result:", result)
print("Available:", ba.available)
print("Allocation:", ba.allocation)
print("Need:", ba.need)
```