

**操作系统实验报告**

|  |  |
| --- | --- |
| **学 院：** | **计算机科学学院** |
| **专 业：** | **计算机科学与技术（创新实验班）** |
| **学 号：** | **42112255** |
| **姓 名：** | **何佳民** |
| **指导教师：** | **孙增国** |

**2023 年 11 月 25日**

**实验三 预防进程死锁的银行家算法**

**一、实验目的**

通过这次实验，加深对进程死锁的理解，进一步掌握进程资源的分配、死锁的检测和安全序列的生成方法。

**二、实验器材**

微机

**三、实验要求**

1）上机前认真复习银行家算法，熟悉资源分配和安全检查过程；

2）上机时独立编程、调试程序；

3）根据具体实验要求，完成好实验报告（包括实验的目的、内容、要求、源程序、实例运行结果截图、发现的问题以及解决方法）。

**四、实验内容**

**1．问题描述：**

设计程序模拟预防进程死锁的银行家算法的工作过程。假设系统中有n个进程P1, … ,Pn，有m类可分配的资源R1, … ,Rm，在T0时刻，进程Pi分配到的j类资源为Allocationij个，它还需要j类资源Need ij个，系统目前剩余j类资源Workj个，现采用银行家算法进行进程资源分配预防死锁的发生。

**2．程序要求：**

1）判断当前状态是否安全，如果安全给出安全序列；如果不安全给出理由。

2）对于下一个时刻T1，某个进程Pk会提出请求Request(R1, … ,Rm)，判断分配给P k进程请求的资源之后系统是否安全。

3）输入：进程个数n，资源种类m，T0时刻各个进程的资源分配情况（可以运行输入，也可以在程序中设置）；

4）输出：如果安全，输出安全的进程序列，不安全则提示信息。

**3．问题分析：**

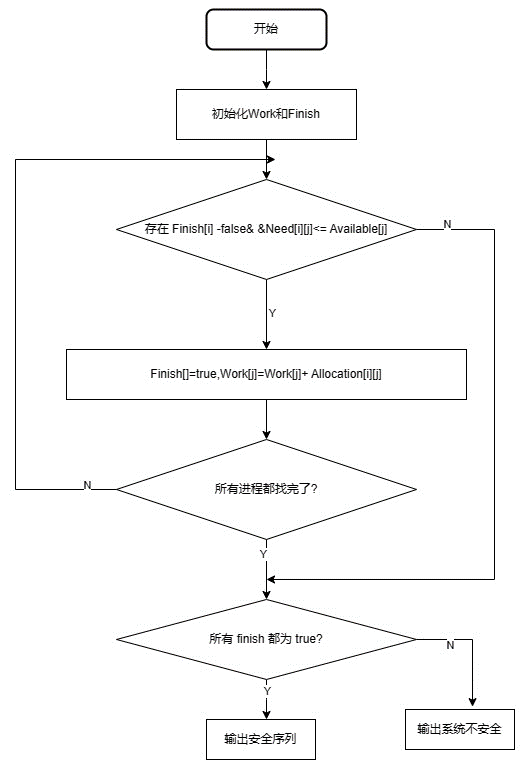


图 3-1 银行家算法流程图

如流程图所示，**银行家算法**如下：

设Request i是进程Pi的请求向量，如果Request i[j]=K，表示进程P i需要K个R j类型的资源。当Pi发出资源请求后，系统按下述步骤进行检查：

1）如果Request i[j]≤Need[i,j]，便转向步骤(2)；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

2）如果Requesti[j]≤Available[j]，便转向步骤(3)；否则，表示尚无足够资源，Pi须等待。

3）系统试探着把资源分配给进程Pi，并修改下面数据结构中的数值：

4）系统执行安全性算法，检查此次资源分配后系统是否处于安全状态。若安全，才正式将资源分配给进程Pi，以完成本次分配；否则，将本次的试探分配作废，恢复原来的资源分配状态，让进程Pi等待。

Available[j]:= Available[j]-Requesti[j]；

Allocation[i,j]:= Allocation[i,j]+Requesti[j]；

Need[i,j]:= Need[i,j]-Request i[j]；

**安全性算法：**

系统所执行的安全性算法可描述如下：

1）设置两个向量：第一个是工作向量Work，它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时，Work:=Available。第二个是Finish，它表示系统是否有足够的资源分配给进程，使之运行完成。开始时先做Finish[i]:=false；当有足够资源分配给进程时，再令Finish[i]:=true。

2）从进程集合中找到一个能满足下述条件的进程：第一【Finish[i]=false；】第二【Need[i,j]≤Work[j]；若找到，执行步骤3)，否则，执行步骤4)】。

3）当进程Pi获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：【Work[j]:= Work[j]+Allocation[i,j]；】【Finish[i]:=true；】【go to step (2)；】

4）如果所有进程的Finish[i]=true都满足，则表示系统处于安全状态；否则，系统处于不安全状态。

**4.源程序：**

Python代码（Python版本3.9）：  
import sys

# 定义常量

M = 3

N = 5

# 初始化各种资源和状态数组

Resource = [0] \* M

Max = [[0] \* M for \_ in range(N)]

Allocation = [[0] \* M for \_ in range(N)]

Need = [[0] \* M for \_ in range(N)]

Available = [0] \* M

Work = [0] \* M

Finish = [0] \* N

List = [0] \* N # 存放安全序列的下标序列

def initial():

""" 创建初始状态：输入 Resource、Max 和 Allocation，计算出 Need 和 Available """

print("Resource--输入M种资源的总数量（用空格分隔）:")

Resource[:] = list(map(int, input().split()))

if len(Resource) != M:

print("资源数量不正确，应为", M, "种资源")

sys.exit(1)

print("Max--输入N个进程分别对M种资源的最大需求量（每个进程的需求量用空格分隔，每个进程占一行）:")

for j in range(N):

Max[j][:] = list(map(int, input().split()))

if len(Max[j]) != M:

print("进程", j, "的资源需求量不正确，应为", M, "种资源")

sys.exit(1)

print("Allocation--输入N个进程获得M种资源的数量（每个进程的资源数量用空格分隔，每个进程占一行）:")

for j in range(N):

Allocation[j][:] = list(map(int, input().split()))

if len(Allocation[j]) != M:

print("进程", j, "的资源分配不正确，应为", M, "种资源")

sys.exit(1)

# 计算 Need 和 Available

for j in range(N):

for i in range(M):

Need[j][i] = Max[j][i] - Allocation[j][i]

for i in range(M):

Available[i] = Resource[i] - sum(Allocation[j][i] for j in range(N))

def printState():

""" 输出当前的状态表 """

print("状态表:\n|Process |Max |Allocation |Need |Available |")

for i in range(N):

if i == 0:

print("|P%-11d|%4d%4d%4d|%4d%4d%4d|%4d%4d%4d|%4d%4d%4d|" % (i, \*Max[i], \*Allocation[i], \*Need[i], \*Available))

else:

print("|P%-11d|%4d%4d%4d|%4d%4d%4d|%4d%4d%4d| |" % (i, \*Max[i], \*Allocation[i], \*Need[i]))

def isfinish():

""" 检查是否存在满足条件的进程，并更新 Finish 和 Work """

for i in range(N):

if Finish[i] == 0 and all(Need[i][j] <= Work[j] for j in range(M)):

for j in range(M):

Work[j] += Allocation[i][j]

Finish[i] = 1

return i

return -1

def issafe():

""" 判断当前状态是否安全，并计算安全序列 """

global Work

Work = Available.copy()

for i in range(N):

Finish[i] = 0

for i in range(N):

a = isfinish()

if a != -1:

List[i] = a

else:

return False

return True

def printList():

""" 输出安全序列 """

print("\n安全序列表如下:\n|Process |Work |Need |Allocation |Work+Alloc |Finish |")

Work = Available.copy()

for i in range(N):

print("|P%-11d|%4d%4d%4d|%4d%4d%4d|%4d%4d%4d|%4d%4d%4d|true" % (List[i], \*Work, \*Need[List[i]], \*Allocation[List[i]], \*map(sum, zip(Work, Allocation[List[i]]))))

for j in range(M):

Work[j] += Allocation[List[i]][j]

def reqresource(i, Request):

""" 处理资源请求 """

global Available, Allocation, Need # 添加这一行以引用全局变量

if all(Request[j] <= Need[i][j] for j in range(M)) and all(Request[j] <= Available[j] for j in range(M)):

temp\_Available = Available.copy()

temp\_Allocation = Allocation[i].copy()

temp\_Need = Need[i].copy()

for j in range(M):

Available[j] -= Request[j]

Allocation[i][j] += Request[j]

Need[i][j] -= Request[j]

if not issafe():

print("\n不存在安全序列，不是安全状态。")

Available, Allocation[i], Need[i] = temp\_Available, temp\_Allocation, temp\_Need

else:

print("\n是安全序列分配成功!")

printList()

else:

print("\n尚无足够的资源，第%d个进程堵塞。" % i)

def main():

""" 主函数 """

initial()

printState()

if not issafe():

print("Initial state is unsafe!")

else:

print("\nInitial state is safe!")

printList()

reqid = int(input("Input the id of request process:"))

while 0 <= reqid < N:

print("Input request resources（用空格分隔）:")

req = list(map(int, input().split())) # 修改这一行

if len(req) != M:

print("资源请求不正确，应为", M, "种资源")

continue

reqresource(reqid, req)

printState()

reqid = int(input("Input the id of request process:"))

if \_\_name\_\_ == "\_\_main\_\_":

main()

**5.运行结果与分析：**

由例题对运行结果进行检查：

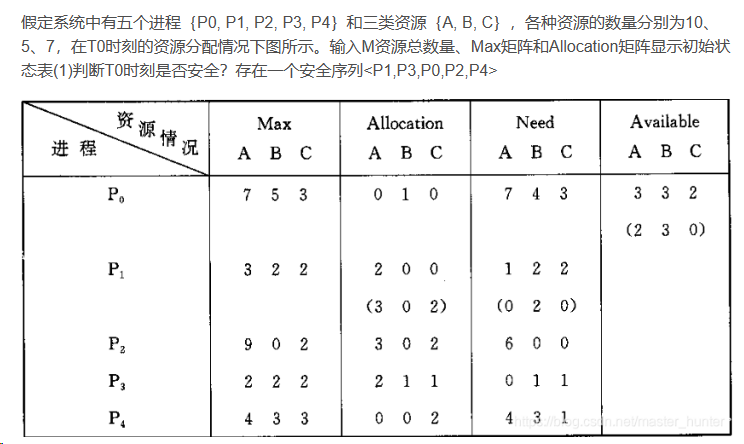


图 3-2 例题

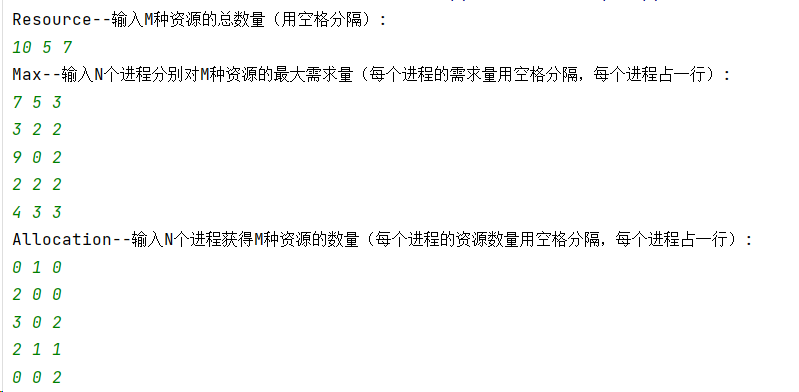
结果输出的界面如下：  


图 3-3 截图

（1）判断T0时刻是否安全？

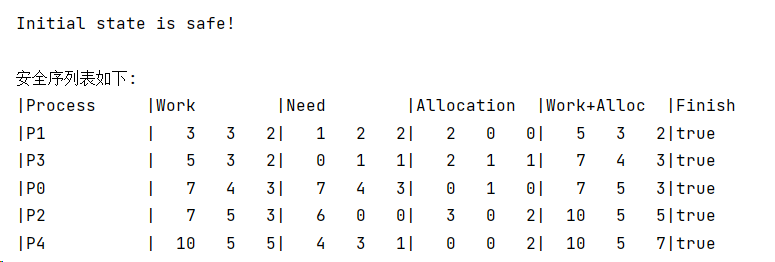


图 3-4 截图

存在这样的安全序列P1,P3,P0,P2,P4。

（2）P1请求资源：P1发出请求向量Request1(1，0，2)。

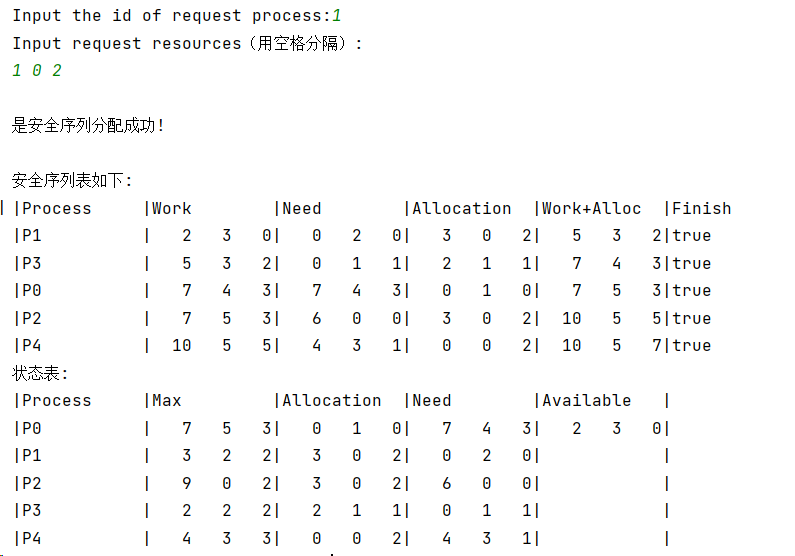


图 3-5 截图

存在安全序列，银行家算法能够给P1分配资源。

（3）在P1的基础上，P4请求资源：P4发出请求向量Request4(3，3，0)。

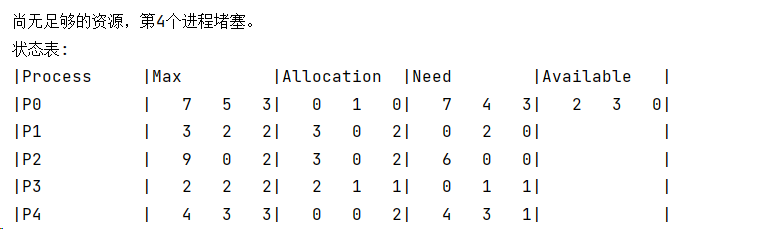


图 3-6 截图

1. Request4(3, 3, 0)≤Need4(4, 3, 1);

② Request4(3, 3, 0) >Available(2, 3, 0)，让P4堵塞等待。状态表没有变化。

(4)P0请求资源：P0发出请求向量Requst0(0，2，0)

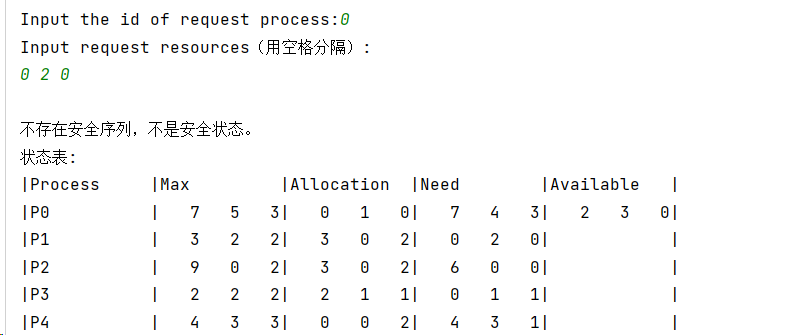


图 3-7 截图

1. Request0(0, 2, 0)≤Need0(7, 4, 3);

② Request0(0, 2, 0)≤Available(2, 3, 0);系统暂时先假定可为P0分配资源，并修改有关数据,如下所示。

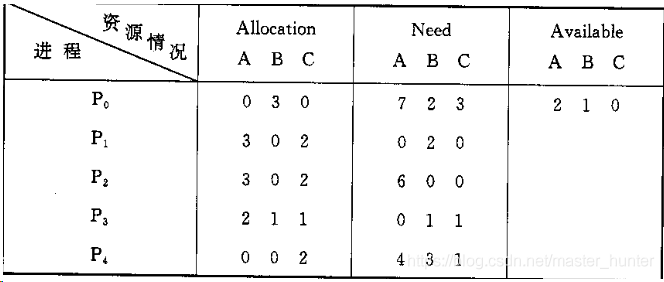


图 3-8 修改数据后的截图

可用资源Available(2,1,0)不能满足任何进程的需求，进入不安全状态。此时系统不分配资源给P0。

（5）若P0发出请求向量Requst0(1，1，0)。

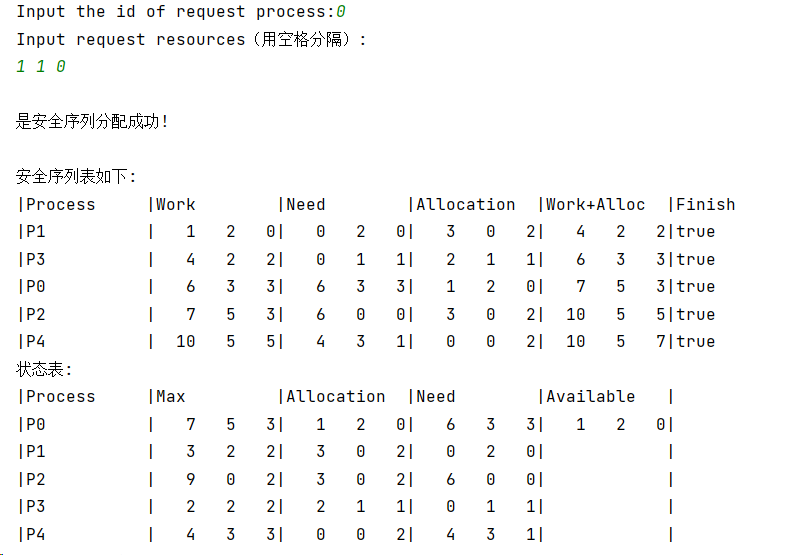


图 3-9 截图

存在安全序列，银行家算法能够给P0分配资源。

综上所述，运行结果正确。

**五、实验总结**

1.

未解析的引用，Available' :87

未解析的引用Available' :88

未解析的引用，Available' :93

解决办法：   
global Available, Allocation, Need # 添加这一行以引用全局变量

2.

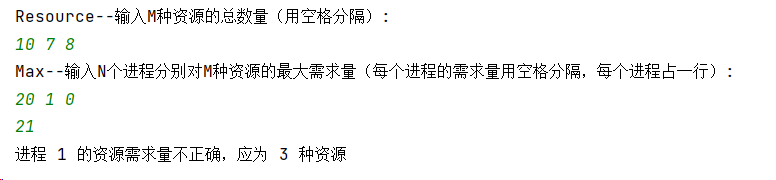


图 3-10 截图

解决办法：添加代码进行检查

3.

ValueError: invalid literal for int() with base 10: '10 5 7'

尝试一次性输入的多个值（'10 5 7'）转换为整数时遇到问题。

解决办法：

在Python中，input()函数每次只能读取一行输入，并将其作为一个字符串返回。如果一行中有多个值，需要分割这个字符串并分别转换每个值。每次输入一行数据，数据之间用空格分隔。例如，当提示输入资源总量时，应该输入类似10 5 7这样的字符串，程序会自动将其分割并转换为整数列表。