第四次作业：死锁

课程名称:	操作系统	指导教师:	张民
姓　　名:	王海生	学　　号:	10235101559

目录

1 问题 7.7

1.1 题目

假设一个系统有 4 个相同类型的资源，并由 3 个进程共享。每个进程最多需要 2 个资源。证明这个系统不会死锁。

1.2 解答

假设系统有 4 个相同类型的资源，且由 3 个进程共享。每个进程最多需要 2 个资源。我们可以设这 3 个进程为 P1、P2 和 P3，每个进程的最大资源需求为 2 个，系统总共有 4 个资源。在任意情况下，无论进程的执行顺序如何，都能够保证每个进程请求的资源数（2 个）始终小于或等于当前可用资源与之前进程已占用资源的

和。因为每个进程最多占用 2 个资源，总资源数为 4，这意味着在任何时刻，总会有足够的资源保证至少一个进程能够完成其操作，并释放资源供其他进程使用。因此，系统始终处于安全状态，必然不会发生死锁。

2 问题 7.12

2.1 题目

假设一个系统具有如下的快照：

	Allocation				Max			
	A	B	C	D	A	B	C	D
P_0	3	0	1	4	5	1	1	7
P_1	2	2	1	0	3	2	1	1
P_2	3	1	2	1	3	3	2	1
P_3	0	5	1	0	4	6	1	2
P_4	4	2	1	2	6	3	2	5

采用银行家算法，确定如下每个状态是否安全的。如果状态是安全的，那么说明进程可以完成的顺序。否则，说明为什么状态是不安全的。

a. Available = (0, 3, 0, 1)

b. Available = (1, 0, 0, 2)

2.2 解答

根据所学的知识：

$$\text{need} = \text{max} - \text{allocation}$$

$$\text{need} = \begin{pmatrix} 2 & 2 & 1 & 1 \\ 2 & 1 & 3 & 1 \\ 0 & 2 & 1 & 3 \\ 0 & 1 & 1 & 2 \end{pmatrix}$$

2.2.1 a 小问

1. 计算需求矩阵（need = max - allocation）：

	A	B	C	D
P_0	2	1	0	3
P_1	1	0	0	1
P_2	0	2	0	0
P_3	4	1	0	2
P_4	2	1	1	3

2. 初始化 `work = available = (0, 3, 0, 1)`，`finish = (0, 0, 0, 0, 0)`，表示所有进程都未完成。

3. 依次检查进程是否能完成：

   (a) 第一步：检查进程 P_2，其需求为 $(0,2,0,0)$，满足 `need[i] <= work`，即 $(0,2,0,0) \leq (0,3,0,1)$，因此可以完成 P_2。

   - 更新 `work = work + allocation[2] = (0, 3, 0, 1) + (3, 1, 2, 1) = (3, 4, 2, 2)`，更新 `finish = (0, 0, 1, 0, 0)`。

   (b) 第二步：检查进程 P_1，其需求为 $(1,0,0,1)$，满足 `need[i] <= work`，即 $(1,0,0,1) \leq (3,4,2,2)$，因此可以完成 P_1。

   - 更新 `work = work + allocation[1] = (3, 4, 2, 2) + (2, 2, 1, 0) = (5, 6, 3, 2)`，更新 `finish = (0, 1, 1, 0, 0)`。

   (c) 第三步：检查进程 P_3，其需求为 $(4,1,0,2)$，满足 `need[i] <= work`，即 $(4,1,0,2) \leq (5,6,3,2)$，因此可以完成 P_3。

   - 更新 `work = work + allocation[3] = (5, 6, 3, 2) + (0, 5, 1, 0) = (5, 11, 4, 2)`，更新 `finish = (0, 1, 1, 1, 0)`。

   (d) 第四步：检查进程 P_0，其需求为 $(2,1,0,3)$，满足 `need[i] <= work`，即 $(2,1,0,3) \leq (5,11,4,2)$，因此可以完成 P_0。

   - 更新 `work = work + allocation[0] = (5, 11, 4, 2) + (3, 0, 1, 4) = (8, 11, 5, 6)`，更新 `finish = (1, 1, 1, 1, 0)`。

   (e) 第五步：检查进程 P_4，其需求为 $(2,1,1,3)$，不满足 `need[i] <= work`，即 $(2,1,1,3) \not\leq (8,11,5,6)$，因此无法完成 P_4。

4. 由于在迭代结束时，`finish` 不全为 true，系统处于非安全状态。对 P_4 的请求可能导致死锁，因此该系统是非安全的。

2.2.2 b 小问

1. 计算需求矩阵（need = max - allocation）：

	A	B	C	D
P_0	2	1	0	3
P_1	1	0	0	1
P_2	0	2	0	0
P_3	4	1	0	2
P_4	2	1	1	3

2. 初始化 `work = available = (1, 0, 0, 2)`，`finish = (0, 0, 0, 0, 0)`，表示所有进程都未完成。

3. 依次检查进程是否能完成：

   (a) 第一步：检查进程 P_1，其需求为 $(1,0,0,1)$，满足 `need[i] <= work`，即 $(1,0,0,1) \leq (1,0,0,2)$，因此可以完成 P_1。

   - 更新 `work = work + allocation[1] = (1, 0, 0, 2) + (2, 2, 1, 0) = (3, 2, 1, 2)`，更新 `finish = (0, 1, 0, 0, 0)`。

   (b) 第二步：检查进程 P_0，其需求为 $(2,1,0,3)$，满足 `need[i] <= work`，即 $(2,1,0,3) \leq (3,2,1,2)$，因此可以完成 P_0。

   - 更新 `work = work + allocation[0] = (3, 2, 1, 2) + (3, 0, 1, 4) = (6, 2, 2, 6)`，更新 `finish = (1, 1, 0, 0, 0)`。

   (c) 第三步：检查进程 P_2，其需求为 $(0,2,0,0)$，满足 `need[i] <= work`，即 $(0,2,0,0) \leq (6,2,2,6)$，因此可以完成 P_2。

   - 更新 `work = work + allocation[2] = (6, 2, 2, 6) + (3, 1, 2, 1) = (9, 3, 4, 7)`，更新 `finish = (1, 1, 1, 0, 0)`。

   (d) 第四步：检查进程 P_3，其需求为 $(4,1,0,2)$，满足 `need[i] <= work`，即 $(4,1,0,2) \leq (9,3,4,7)$，因此可以完成 P_3。

   - 更新 `work = work + allocation[3] = (9, 3, 4, 7) + (0, 5, 1, 0) = (9, 8, 5, 7)`，更新 `finish = (1, 1, 1, 1, 0)`。

   (e) 第五步：检查进程 P_4，其需求为 $(2,1,1,3)$，满足 `need[i] <= work`，即 $(2,1,1,3) \leq (9,8,5,7)$，因此可以完成 P_4。

   - 更新 `work = work + allocation[4] = (9, 8, 5, 7) + (4, 2, 1, 2) = (13, 10, 6, 9)`，更新 `finish = (1, 1, 1, 1, 1)`。

   (f) 所有进程都能够完成，`finish` 全部为 true，因此该系统是安全的。

安全序列： P_1, P_0, P_2, P_3, P_4

3 问题 7.13

3.1 题目

假设一个系统具有如下的快照：

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	0	1	4	2	1	2				
P_1	3	1	2	1	5	2	5	2				
P_2	2	1	0	3	2	3	1	6				
P_3	1	3	1	2	1	4	2	4				
P_4	1	4	3	2	3	6	6	5				

采用银行家算法，回答下面的问题：a. 通过进程可以完成执行的顺序，说明系统处于安全状态。b. 当进程 P_1 的请求为 $(1,1,0,0)$ 时，能否立即允许这一请求？c. 当进程 P_4 的请求为 $(0,0,2,0)$ 时，能否立即允许这一请求？

3.2 解答

3.2.1 a 小问

1. 初始化 `work = available = (3, 3, 2, 1)`，`finish = (0, 0, 0, 0, 0)`，表示所有进程都未完成。
2. 依次检查进程是否能完成：
   (a) 第一步：检查进程 P_0，其需求为 `need[0]`，满足 `need[0] <= work`，因此可以完成 P_0。
   - 更新 `work = work + allocation[0] = (3, 3, 2, 1) + (2, 0, 0, 1) = (5, 3, 2, 2)`，更新 `finish = (1, 0, 0, 0, 0)`。

   (b) 第二步：检查进程 P_3，其需求为 `need[3]`，满足 `need[3] <= work`，因此可以完成 P_3。
   - 更新 `work = work + allocation[3] = (5, 3, 2, 2) + (1, 3, 1, 2) = (6, 6, 3, 4)`，更新 `finish = (1, 0, 0, 1, 0)`。

   (c) 第三步：检查进程 P_1，其需求为 `need[1]`，满足 `need[1] <= work`，因此可以完成 P_1。
   - 更新 `work = work + allocation[1] = (6, 6, 3, 4) + (3, 1, 2, 1) = (9, 7, 5, 5)`，更新 `finish = (1, 1, 0, 1, 0)`。

   (d) 第四步：检查进程 P_2，其需求为 `need[2]`，满足 `need[2] <= work`，因此可以完成 P_2。

- 更新 `work = work + allocation[2] = (9, 7, 5, 5) + (2, 1, 0, 3) = (11, 8, 5, 8)`，更新 `finish = (1, 1, 1, 1, 0)`。

(e) 第五步：检查进程 P_4，其需求为 `need[4]`，满足 `need[4] <= work`，因此可以完成 P_4。

- 更新 `work = work + allocation[4] = (11, 8, 5, 8) + (1, 4, 3, 2) = (12, 12, 8, 10)`，更新 `finish = (1, 1, 1, 1, 1)`。

(f) 迭代结束，`finish` 对所有进程均为 true，系统处于安全状态。一个可完成的进程执行顺序为 **<P0, P3, P1, P2, P4>**。

3.2.2 b 小问

1. 检查进程 P_1 的请求 `request[1] = (1, 1, 0, 0)`，满足 `request[1] <= need[1]` 和 `request[1] <= available`。
   - 更新 `available = available - request[1] = (2, 2, 2, 1)`。
   - 更新 `allocation[1] = allocation[1] + request[1] = (4, 2, 2, 1)`。
   - 更新 `need[1] = need[1] - request[1] = (1, 0, 3, 1)`。
2. 修改状态后，经检验系统仍为安全状态，则请求可被立刻允许。

3.2.3 c 小问

1. 检查进程 P_4 的请求 `request[4] = (0, 0, 2, 0)`，满足 `request[4] <= need[4]` 和 `request[4] <= available`。
   - 更新 `available = available - request[4] = (3, 3, 0, 1)`。
   - 更新 `allocation[4] = allocation[4] + request[4] = (1, 4, 5, 2)`。
   - 更新 `need[4] = need[4] - request[4] = (2, 2, 1, 3)`。
2. 修改状态后,经检验系统为非安全状态（安全算法第一步就无法找到 i 使得 `need[i] <= work = available`），则该请求不能立刻被允许，进程 P_4 应该等待该请求且系统恢复到原来的资源分配状态。