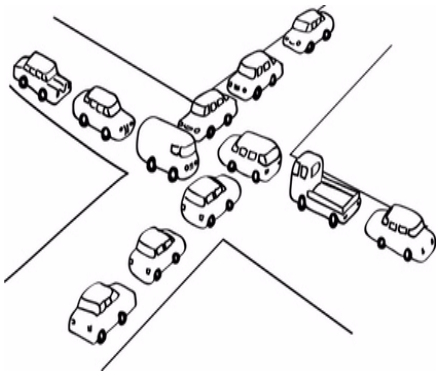
Prof. Dr. Burkhard Stiller, University of Zurich, Binzmühlestrasse 14, CH-8050 Zürich
Telephone: +41 44 635 6710, Fax: +41 44 635 6809, stiller@ifi.uzh.ch
Bruno Rodrigues, Telephone: +41 44 635 4381, rodrigues@ifi.uzh.ch
Sina Rafati, Telephone: +41 44 635 6702, rafati@ifi.uzh.ch

Detection Avoidance - Banker's Algorithm

Hand-out date: Mar 25, 2020
Submission until and discussion: --



1. Banker's Algorithm

When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. We need the following data structures, where ***n*** is the number of processes in the system and ***m*** is the number of resource types:

- **Available**: a vector of lenght ***m*** indicates the number of available resources of each type. If *available[]* equals *k*, then *k* instances of resource type *Rj* are available.

- **Max**: An ***n x m*** matrix defines the maximum demand of each process. If *Max[i][j]* equals *k*, then process *Pi* may request at most *k* instances of resource *Rj*.

- **Allocation**: An ***n x m*** matrix defines the number of resources of each type currently allocated to each process. If *Allocation[i][j]* equals *k*, then process *Pi* is currently allocated *k* instances of resource type *Rj*.

- **Need**: An **n** x **m** matrix indicates the remaining resource need of each process. If *Need[i][j]* equals *k*, then process *Pi* may need *k* more instances of resource type *Rj* to complete its task. Note that:

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

2. Safety Algorithm

Consider ***Work*** and ***Finish*** as temporary vectors of lenght *m* and *n*:

Work [], Finish []

1 - Initialize
Work = Available
Finish[i] = false for i = 0, 1, ..., N
2 - Find and index *i* such that both:
a) *Finish[i] == false*
b) *Need[i] <= Work*
If no such i exists, go to step 4.
3 - Work = Work + Allocation;
Finish[i] = true
Go to step 2.
4 - If *Finish[i]* == *true* for all *i*, then the system is in a safe state.

3. Resource-Request Algorithm

Let *Request* be the request vector for process *Pi*. If *Request* [j] == k, then process *Pi* wants *k* instances of resource type *Rj*. When a request for resources is made by process *Pi*, the following actions are taken:

1. **If *Request* <= *Need***, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2. **If *Request* <= *Available***, go to step 3. Otherwise, *Pi* must wait, since the resources are not available.

3. Have the system pretend to have allocated the requested resources to process *Pi* by modifying the state as follows:

Available = Available - Request
Allocation =Allocation; + Request
Need = Need - Request

4. If the resulting resource-allocation state is ***safe***, the transaction is completed, and process *Pi* is allocated its resources. However, if the new state is ***unsafe***, then *Pi* must wait for Request, and the old resource-allocation state is restored.

4. Applying the Banker's algorithm - Guided Example from Slides

Consider a system with five processes P0 through P4 and three resources types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

1. Calculate the content of the *Need* matrix is:

Formula: *Need[i][j] = Max[i][j] - Allocation[i][j])*

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Applying the formula step-by-step:

P0 = (7-0), (5-1), (3-0) = **[7, 4, 3]**
P1 = (3-2), (2-0), (2-0) = **[1, 2, 2]**
P2 = (9-3), (0-0), (2-2) = **[6, 0, 0]**
P3 = (2-2), (2-1), (2-1) = **[0, 1, 1]**
P4 = (4-0), (3-0), (3-2) = **[4, 3, 1]**

2. To check whether the system is in a safe state apply the *Safety Algorithm*:

[STEP 1] // Initialization
m = resources (A, B and C) = **3**
n = processes (P0, P1, P2, P3, P4) = **5**
Work = Available Resources = **[3, 3, 2]**
Finish[i] = [P0, P1, P2, P3, P4] = **[false, false, false, false, false]**

[STEP 2] // Find an "i" that is marked as "*false*" and satisfies *Need* **<=** *Work.*
i=0
- Need0 = [7, 4, 3]
- Work = [3, 3, 2]
- Need >= Work, P0 must wait.

[STEP 2]
i=1
- Need1 = [1, 2, 2]
- Work = [3, 3, 2]
- Need <= Work, P1 must be in the safe sequence (safe = [P1])

[STEP 3]
Work = Work + Allocation = [3, 3, 2] + [2, 0, 0] = [(3+2), (3+0), (2+0)] = **[5, 3, 2]**
Finish = **[false, true, false, false, false]**

[STEP 2]
i=2
- Need2 = [6, 0, 0]
- Work = [5, 3, 2]
- Need >= Work, P2 must wait.

[STEP 2]
i=3
- Need3 = [0, 1, 1]
- Work = [5, 3, 2]

- Need <= Work, P3 must be in the safe sequence (safe = [P1, P3])

[STEP 3]
Work = Work + Allocation = [5, 3, 2] + [2, 1, 1] = [(5+2), (3+1), (2+1)] = **[7, 4, 3]**
Finish = **[false, true, false, true, false]**

[STEP 2]
i=4
- Need4 = [4, 3, 1]
- Work = [7, 4, 3]
- Need <= Work, P4 must be in the safe sequence (safe = [P1, P3, P4])

[STEP 3]
Work = Work + Allocation = [7, 4, 3] + [0, 0, 2] = [(7+0), (4+0), (3+2)] = **[7, 4, 5]**
Finish = **[false, true, false, true, true]**

[STEP 2]
i=2 // comes back from i=4 and find i=2 as "false"
- Need2 = [6, 0, 0]
- Work = [7, 4, 5]
- Need <= Work, P2 must be in the safe sequence (safe = [P1, P3, P4, P2])

[STEP 3]
Work = Work + Allocation = [7, 4, 5] + [3, 0, 2] = [(7+3), (4+0), (5+2)] = **[10, 4, 7]**
Finish = **[false, true, true, true, true]**

[STEP 2]
i=0
- Need0 = [7, 4, 3]
- Work = [10, 4, 7]
- Need <= Work, P0 must be in the safe sequence (safe = [P1, P3, P4, P2, P0])

[STEP 3]
Work = Work + Allocation = [10, 4, 7] + [0, 1, 0] = [(10+0), (4+1), (7+0)] = **[10, 5, 7]**
Finish = **[true, true, true, true, true]**

[STEP 2]
//No more Finish[i] == false :)

[STEP 4]
Finish = [true, true, true, true, true]
System is in a safe state, safe sequence = [P1, P3, P4, P2, P0]