Banker's Algorithm Problem Set

Banker's Algorithm Review

Let's first review the Banker's Algorithm, which is a deadlock avoidance algorithm. It is called the Banker's Algorithm, because it could be used by a bank to make sure that money is allocated in such a way that all customer needs are met.

When a new process enters the system, it declares the maximum number of instances that are needed. This number cannot exceed the total number of resources in the system. If the process can be accommodated based upon the needs of the system, then resources are allocated, otherwise the process must wait. The algorithm is actually made up of two separate algorithms: the safety algorithm and the resource allocation algorithm.

The following data structures are needed:

- n represents the number of processes and m represents the number of resource types.
- Available
    1. A vector (array) of available resources of each type
    2. If available[j] = k, then k instances of R_j are available.
- Max
    1. An n by m matrix
    2. Defines maximum demand for each process
    3. Max[i][j] = k, then process P_i may request at most k instances of resource R_j.
- Allocation
    1. An n by m matrix
    2. Defines number of resources of each type currently allocated to each process
    3. Allocation[i][j]=k, then process P_i is currently allocated k instances of R_j.
- Need
    1. An n by m matrix
    2. Indicates remaining resource need of each process
    3. If need[i][j] = k, then process P_i needs k more instances of R_j.
    4. Need[i][j]= Max[i][j]-allocation[i][j]

A few things about notation:

1. Let X and Y be vectors of length n. X≤Y if and only if X[i]≤Y[i] for all I =1,2,…n
2. Example
   - X = (1,7,3,2) Y =(0,3,2,1) then Y≤X
3. We can treat each row in allocation and need as vectors and call them $allocation_i$ and $need_i$.

The safety algorithm is used to determine if a system is in a safe state. It works as follows:

1. Work and finish: vectors of length m and n respectively
   Initialize work = available
   Finish[i] = false for I = 0,1,…,n-1

2. Find an 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 + $allocation_i$ Finish[i] = true.
   Go to step 2.

4. If all finish[i] = true, then the system is in a safe state.

The resource-request algorithm is used to determine whether requests can be safely granted. It works as follows:

- Let $request_i$ be the request vector for process Pi. If $request_i[j] = k$, then process P_i wants k instances of resources R_j.
- When a request is made by process Pi, the following actions are taken:
  1. If $request_i$ ≤need go to step 2; otherwise, raise an error condition.
  2. If $request_i$ ≤available, go to step 3, otherwise Pi must wait since resources are not yet available.
  3. Have system pretend to allocate the requested resources to process Pi by modifying the state as follows:
     Available = available - $request_i$
     $Allocation_i$ = $allocation_i$ -+ $request_i$

If the resulting resource allocation state is safe, then transaction is complete and P_i is allocated its resources. If unsafe, P_i must wait and old state is restored.

Example

In the following example, I have omitted all of the steps of the safety algorithm. If you would like to see this completely worked out, please refer to the Banker's Algorithm video note.

5 processes $P_0 \rightarrow P_4$
3 resources A B C
At time T_o, a snapshot of the system has been taken:

	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	2			

Need is defined as max – allocation

	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

First, we should determine if this is a safe state.

Define two vectors Work (3,3,2) and Finish (false,false,false,false,false).

After we work through the algorithm to check for safe state, we will see that P_1,P_3,P_4,P_2,P_0 is a safe state.
Now, P_1 requests an additional instance of resource type A and 2 of type C.
$Request_1 = (1,0,2)$
Is request1 ≤ available?
(1,0,2) ≤ (3,3,2)
Next, let's pretend that request has been granted. We arrive at a new state:

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

Now, execute the safety algorithm. We will find that $<P_1,P_3,P_4,P_0,P_2>$ is in a safe state.

If P_4 now requests (3,0,0), this cannot be granted, because resources are not available.

Now, let's assume that P_0 requests (0,2,0). Resources are available. This will result in the following state:

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

Is this in a safe state? Work is defined as (2,1,0). We can see that none have a need ≤ work. Therefore, this is in an unsafe state.

Exercise 1

Assume that there are 5 processes, P_0 through P_4, and 4 types of resources. At T_0 we have the following system state:

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

1. Create the need matrix (max-allocation)
2. Use the safety algorithm to test if the system is in a safe state.
3. If the system is in a safe state, can the following requests be granted, why or why not? Please also run the safety algorithm on each request as necessary.
    a. P1 requests (2,1,1,0)
    b. P1 requests (0,2,1,0)

Please check your answers against the answer key.

Exercise 2

Assume that there are three resources, A, B, and C. There are 4 processes P_0 to P_3. At T_0 we have the following snapshot of the system:

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

1. Create the need matrix.
2. Is the system in a safe state? Why or why not?

Please check your answers against the answer key.