Avoidance Algorithms

- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the banker's algorithm

Banker’s Algorithm

- Multiple instances
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

- **Available**: Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available
- **Max**: n x m matrix. If $Max\ [i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation**: n x m matrix. If Allocation$[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need**: n x m matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

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

Safety Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:

   Work = Available

   ***Finish [i] = false* for *i* = 0, 1, …, *n*- 1**

2. Find an ***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 ***Finish [i] == true*** for all ***i***, then the system is in a safe state

Resource-Request Algorithm for Process P_i

$\boldsymbol{Request_i}$ = request vector for process $\boldsymbol{P_i}$. If $\boldsymbol{Request_i\,[j] = k}$ then process $\boldsymbol{P_i}$ wants $\boldsymbol{k}$ instances of resource type $\boldsymbol{R_j}$

1. If $\boldsymbol{Request_i \le Need_i}$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $\boldsymbol{Request_i \le Available}$, go to step 3. Otherwise $\boldsymbol{P_i}$ must wait, since resources are not available
3. Pretend to allocate requested resources to $\boldsymbol{P_i}$ by modifying the state as follows:

$$\boldsymbol{Available = Available - Request_i;}$$

$$\boldsymbol{Allocation_i = Allocation_i + Request_i;}$$

$$\boldsymbol{Need_i = Need_i - Request_i;}$$

- If safe $\Rightarrow$ the resources are allocated to $\boldsymbol{P_i}$
- If unsafe $\Rightarrow$ $\boldsymbol{P_i}$ must wait, and the old resource-allocation state is restored

Example of Banker's Algorithm

- 5 processes P_0 through P_4;
  3 resource types:
  A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time T_0:

	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	

Example (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

	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

- The system is in a safe state since the sequence $< P_1, P_3, P_4, P_2, P_0>$ satisfies safety criteria

Example: P_1 Request (1,0,2)

Check that Request ≤ Need (that is, (1,0,2) ≤ (1,2,2) ⇒ true

Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ true

Updated

	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	

Available = Available – Request$_i$;

Allocation$_i$ = Allocation$_i$ + Request$_i$;

Need$_i$ = Need$_i$ – Request$_i$;

Executing safety algorithm shows that sequence < P_1, P_3, P_4, P_0, P_2> satisfies safety requirement

- Q. Can request for (3,3,0) by P_4 be granted immediately?
- Q. Can request for (0,2,0) by P_0 be granted immediately?

Example: P_4 Request (3,3,0)

Check that Request ≤ Need (that is, (3,3,0) ≤ (4,3,1) ⇒ true

Check that Request ≤ Available (that is, (3,3,0) ≤ (2,3,0) ⇒ false

Request can not immediately be granted. P4 has to wait since resources are not available.

Example: P_0 Request (0,2,0)

Check that Request ≤ Need (that is, (0,2,0) ≤ (7,4,3) ⇒ true

Check that Request ≤ Available (that is, (0,2,0) ≤ (2,3,0) ⇒ true

Updated

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 3 0	7 2 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	

Unsafe.

P0 must wait. The old resource-allocation state will restore.

Final restored 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	

Q. Assume no. of process =4, Max request per process is = 3, what is the minimum no. of resources that will not lead to deadlock condition.

A. 4(3-1)+1 = 9, minimum 9 resource

Q. Consider a system consisting of **m** resources of the same type being shared by **n** processes. Resources can be requested and released by processes only one at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between 1 and m resources.

b. The sum of all maximum needs is less than m+n.

Q. Suppose n processes, P1, …. Pn share m identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process Pi is Si, where Si > 0. Which one of the following is a sufficient condition for ensuring that deadlock does not occur?

(a) $\forall i, s_i < m$ (b) $\forall i, s_i < n$

(c) $\sum_{i=1}^{n} s_i < (m+n)$ (d) $\sum_{i=1}^{n} s_i < (m * n)$

A. option C

Solution

$P_1 \quad P_2 \quad P_3 \quad P_4 \ \ldots\ldots\ldots\ldots\ P_n$

$S_1 \quad S_2 \quad S_3 \quad S_4 \ \ldots\ldots\ldots\ldots\ S_n$

$(S_1 - 1) + (S_2 - 1) + (S_3 - 1) + (S_4 - 1) + \ldots\ldots\ldots\ldots (S_n - 1) + 1 = m$

$(S_1 + S_2 + S_3 + S_4 + \ldots S_n) + (1 + 1 + 1 + 1 + \ldots . n) + 1 = m$

$\sum^n S_i + n = m$

$\sum^n S_i < m+n$