

Deadlock avoidance using Banker's algorithm

S.Rajarajan
CSE/SASTRA





Banker's algorithm

- Usage in banks
 - Clients are asking for over-draft to run their businesses up to an agreed limit based on their eligibility as determined by the bank.
 - The banker knows that not all clients need their limit simultaneously.
 - All clients must receive the money up to their total limits at some point of time but not necessarily at once and simultaneously.
 - They will only demand a quota of money that is needed for the current requirement.
 - After fulfilling their needs, the clients will pay-back their loans



- Example: The banker knows that all 4 clients need 22 units together, but he has only total 10 units

State A

Name	Max Limit	Granted so far
Gopal	8	4
Ram	4	2
David	5	1
Rani	3	0
Bank have got		3

State B

Name	Max Limit	Granted so far
Gopal	8	4
Ram	4	2
David	5	2
Rani	3	1
Bank have got		1

State A



- State A is a safe state since after the disbursal of some portion of money to the clients the bank have got 3 rupees.
- With three rupees they can meet the requirement of Ram.
- Then if Ram returns the money bank will have 5.
- Then Gopal's requirement can be satisfied and he would return his money of 8.
- Then the bank will have 9 rupees.
- In this way David and Rani will receive their OD limits and the bank will reclaim the total money of 10 rupees.



State B

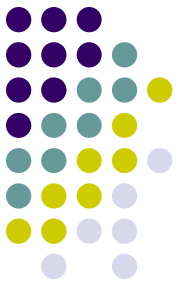
- In state B bank only have 1 rupee after the disbursal of 9 rupees.
- With that sum they can not meet the total OD requirement of any one of their clients.
- Then all the clients will be demanding their remaining quota without returning their allotted fund.
- This situation is unsafe since it has deadlocked.
- Bank should avoid reaching this state

Banker's algorithm for Concurrent Processes



- Always keep so many resources that satisfy the needs of at least one client.
- Multiple instances of a resource might be available and a single process may request multiple instances of a resource.
- Each process must on a priori claim maximum use before commencement.
- When a process requests a resource it may have to wait if it is not available or if the OS is not willing to grant the resource due to banker's algorithm decision.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm



- m - no of resource, n - no of processes
- **Resource vector:** Vector of length m . Comprised of the total number of resources available with the system. If Resource $[j] = k$ then k instances of R_j th resource is available.
- **Claim/Max matrix $n \times m$:** Total requirements of each process for every resource. If $\text{Claim}[x,y] = z$ then P_x is going to require at most z instances of resource R_y .
- **Allocation matrix $n \times m$:** Current allocations to various processes of their resource requirements. If $\text{Allocation}[i,j] = k$ then P_i is currently allocated k instances of R_j .



- **Need matrix $n \times m$:** The pending requirements of each process after their partial allocations. If $\text{Need}[i,j] = k$, then P_i may need k more instances of R_j to complete its task.
- $\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$.
- **Available vector:** Current availability of resources of each type after some allocations. If $\text{available}[j] = k$ then k instances of R_j th resource is currently available.

Determining safe state with the help of Banker's algorithm



- Whenever a process request for additional resources, the OS has to carefully evaluate the consequences of the request being granted before deciding whether to Allocate or Block the process.
- It is computed as follows:
 - Check whether the resource (s) are available currently.
 - Verify that the present request of the process does not exceed its total Claim
 - Tentatively allocate the resource/resources to the process

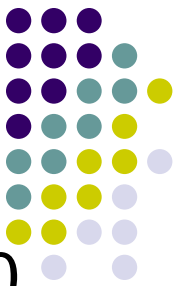


- Modify the Allocation, Need and Available vectors appropriately - **POINT A**
- Repeat the following steps until all the processes are completed and their Allocation and Needs rows have contain zeros.
 - Find a process P_i which can be completed with the help of the Available collection of resources by fulfilling its complete claim for resources as per the Need vector.
 - If no process could be found then return 'Unsafe'
 - Update Allocation and Need vector rows of process P_i by replacing their entries by zero. Update the Available vector by adding the Allocation entries of process P_i into Available entries Available vector
 - $\text{Available } R_i = \text{Allocation } [P_i, R_i]$
- Return 'Safe'



- If return value is safe then reverse all the modifications made to Available, Allocation and Need after POINT A.
- If return value is un-safe then reverse all the modifications made to Available, Allocation and Need up to POINT A. Block the process requested the resource (s).

Example



- 5 processes P0 through P4 ; 3 resource types A (10 instances), B (5 instances, and C (7 instances)
- Snapshot at time T0 :

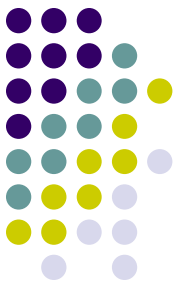
	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Total</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	10 5 7
P_1	2 0 0	3 2 2	1 2 2	<u>Allocated</u>
P_2	3 0 2	9 0 2	6 0 0	7 2 5
P_3	2 1 1	2 2 2	0 1 1	<u>Available</u>
P_4	0 0 2	4 3 3	4 3 1	3 3 2

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



P1 requests resources (1,0,2)

- Now P1 requests 1 instance of R1, 0 instance of R2 and 2 instances of R3
- Step 1: Verify that Request \leq Need that is $(1,0,2) \leq (1,2,2)$
- Step 2: Check that Request \leq Available that is, $(1, 0, 2) \leq (3, 3, 2) = \text{true}$
- Step 3: Run the banker's/ safety algorithm to evaluate the safe/unsafe state of system after the resources are sanctioned to P1.



- Banker's algorithm based evaluation:
 - I. Allocate 1 of R1 and 2 of R3 to P1.
 - II. Available vector becomes (2,3,0)
 - III. Process P1 can be completed with the Available resources (2,3,0) since its requirements of (0,2,0) can be met with available resources.
 - IV. If P1 complete then its Allocation and Need becomes zero. Available will increase as (5,3,2)
 $\Rightarrow (3,2,2) + (2,1,0)$
 - V. Then Process P4 or P3 can be completed.
 - VI. If P3 completes then Available will become (7,4,3)
 $\Rightarrow (5,2,1) + (2,2,2)$
 - VII. P4 can be completed next. Then the Available will become (7,4,5) $\Rightarrow (3,1,2) + (4,3,3)$
 - VIII. Now either P4 or P3 can be completed



Contd..

- ix. If P4 is allocated all its Need then it will complete. Then the available will become $(7,5,5) \Rightarrow (0,1,2) + (7,4,3)$
- x. Finally when P2 completes the Available will become $(10,5,7)$ which is equal to the actual Available resources and all the processes could be completed by making their Need and Allocation to 0.
- xi. Therefore the algorithm determines that it is Safe to allocate $(1,0,2)$ to P1 since there is a safe sequence available (P1, P3, P4, , P0, P2) to complete all processes without deadlock.



- Find whether request for $(3,3,0)$ by P4 be granted.
- Find whether request for $(0,2,0)$ by P0 be granted

Banker's Algorithm – William Stallings



```
struct state {  
    int resource[m];  
    int available[m];  
    int claim[n][m];  
    int alloc[n][m];  
}
```

```
if (alloc [i,*] + request [*] > claim [i,*])  
    <error>;                                /* total request > claim*/  
else if (request [*] > available [*])  
    <suspend process>;  
else {                                       /* simulate alloc */  
    <define newstate by:  
    alloc [i,*] = alloc [i,*] + request [*];  
    available [*] = available [*] - request [*]>;  
}  
if (safe (newstate))  
    <carry out allocation>;  
else {  
    <restore original state>;  
    <suspend process>;  
}
```



```
boolean safe (state S) {  
    int currentavail[m];  
    process rest[<number of processes>];  
    currentavail = available;  
    rest = {all processes};  
    possible = true;  
    while (possible) {  
        <find a process Pk in rest such that  
            claim [k,*] - alloc [k,*] <= currentavail;  
        if (found) {                                /* simulate execution of Pk */  
            currentavail = currentavail + alloc [k,*];  
            rest = rest - {Pk};  
        }  
        else possible = false;  
    }  
    return (rest == null);  
}
```

Additional problem



Given the following state for the Banker's Algorithm.

6 processes P0 through P5

4 resource types: A (15 instances); B (6 instances)

C (9 instances); D (10 instances)

Snapshot at time T0:

Available								
		A	B		C	D		
		6	3		5	4		
Process	Current allocation				Maximum demand			
	A	B	C	D	A	B	C	D
P0	2	0	2	1	9	5	5	5
P1	0	1	1	1	2	2	3	3
P2	4	1	0	2	7	5	4	4
P3	1	0	0	1	3	3	3	2
P4	1	1	0	0	5	2	2	1
P5	1	0	1	1	4	4	4	4

- Verify that the Available array has been calculated correctly.
- Calculate the Need matrix.
- Show that the current state is safe, that is, show a safe sequence of processes. In addition, to the sequence show how the Available (working array) changes as each process terminates.
- Given the request (3,2,3,3) from Process P5. Should this request be granted? Why or why not?