Deadlock

Md. Asifur Rahman
Lecturer
Department of CSE

Deadlock

When a waiting process is not again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a **deadlock**.

Deadlock

Condition for Deadlock:

- Mutual Exclusion
- Hold and Wait
- No preemption
- Circular Wait

Resource Allocation graph

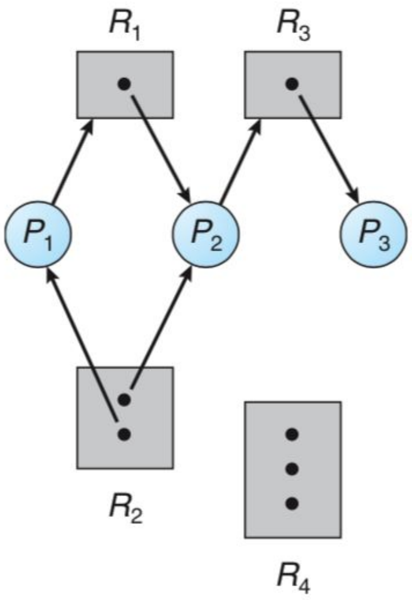


Figure 7.1 Resource-allocation graph.

- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

Resource Allocation graph

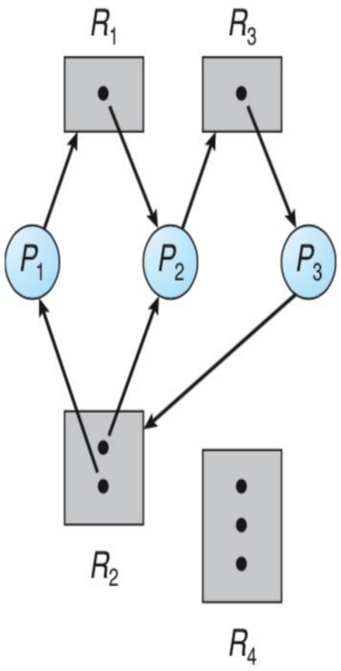


Figure 7.2 Resource-allocation graph with a deadlock.

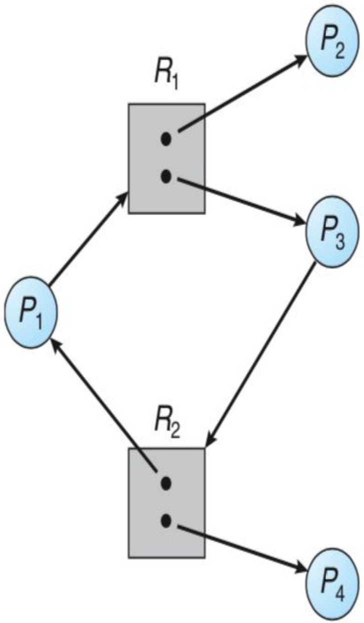


Figure 7.3 Resource-allocation graph with a cycle but no deadlock.

Methods for Deadlock Handling

Deadlock can be dealt with in one of the three ways:

- Use protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.
- Allow the system to enter a deadlocked state,detect it,and finally recover.
- Ignore the problem altogether and pretend that deadlocks never occur in the system.

Methods for Deadlock Handling

- Deadlock Prevention
    - Mutual Exclusion
    - Hold and wait
    - No preemption
    - Circular wait
- Deadlock Avoidance
    - Safe state check using
        - Resource allocation graph algorithm (Single resource instance)
        - Banker’s algorithm ((Multiple resource instance)

Methods for Deadlock Handling

- Deadlock Detection
    - Single Instance of Resource: Detect cycle in **wait for graph**
    - Multiple Instance of Resource: Resource Request Algorithm
        - Recovery From Deadlock
            - Process Termination:
                - Abort all deadlocked processes.
                - Abort one process at a time until the deadlock cycle is eliminated.
            - Resource Preemption:
                - Select victim
                - Rollback
                - Starvation

Resource Allocation graph

Wait-for-graph:

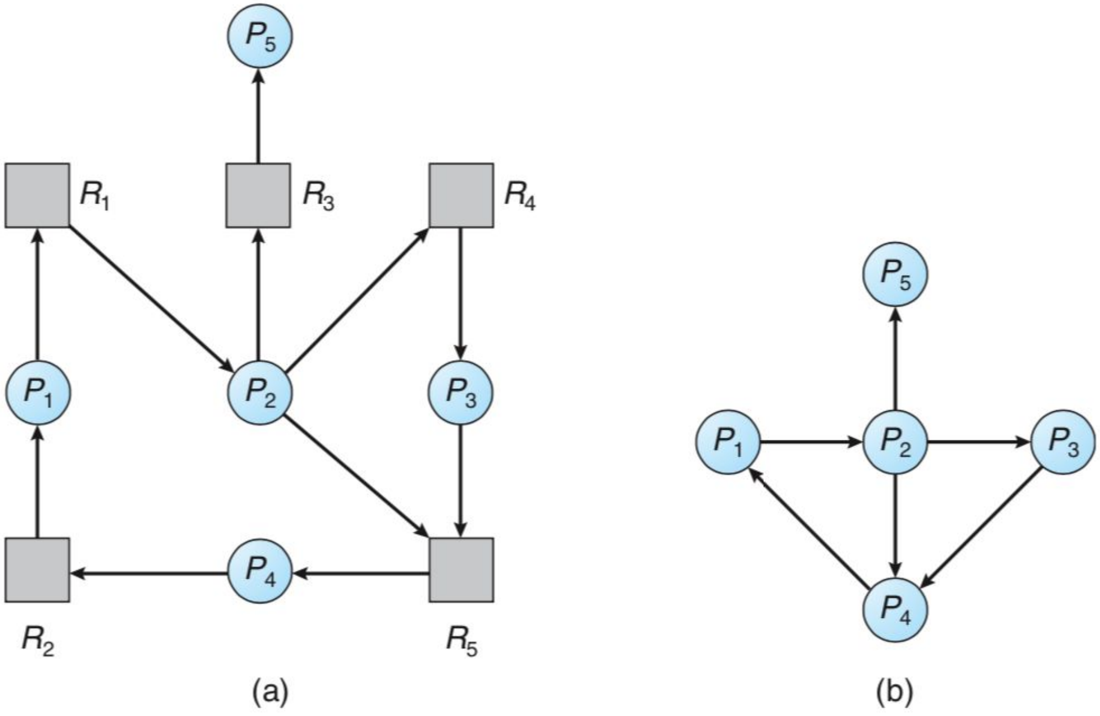


Figure 7.9 (a) Resource-allocation graph. (b) Corresponding wait-for graph.

Banker’s Algorithm

Banker’s Algorithm is a **resource allocation** and **deadlock avoidance** algorithm developed by **dijkstra**.

Banker's Algorithm

Available:	A	B	C
	4	2	3

Available instance of resources

	A	B	C
P1	6	5	4
P2	3	2	2
P3	9	0	2
P4	1	3	5

Max

Denotes maximum resource demand of each process

	A	B	C
P1	1	2	0
P2	1	1	2
P3	2	0	2
P4	1	2	2

Allocation

Denotes number of resource of each type allocated to each process

Banker's Algorithm

Available:	A	B	C
	4	2	3

Available instance of resources

- **Max - Allocation = Need**

	A	B	C
P1	6	5	4
P2	3	2	2
P3	9	0	2
P4	1	3	5

Max

Denotes maximum resource demand of each process

−

	A	B	C
P1	1	2	0
P2	1	1	2
P3	2	0	2
P4	1	2	2

Allocation

Denotes number of resource of each type allocate to each process

=

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

Denotes number of resource of each type required by each process

Safe State

- **Initially:**

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

Denotes number of resource of each type required by each process

	A	B	C
P1	1	2	0
P2	1	1	2
P3	2	0	2
P4	1	2	2

Allocation

Denotes number of resource of each type allocate to each process

	A	B	C
Available:	4	2	3

P1	False
P2	False
P3	False
P4	False

Finish

Safe State

- **Now we will check for a Process P[i] if (Available[j] >= Need[i][j])**
    - **False :**
        - **Do nothing**
        - **Go to next process**
    - **True:**
        - **Available[j] := Available[j] + Allocation[i][j]**
        - **Finish[i]:=true**
        - Keep the need matrix unchanged
        - Keep the allocation matrix unchanged

Safe State

- **Safe state check: (_,**

Need of P1 > Available

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

~~<=~~

	A	B	C
Available:	4	2	3

False
False
False
False

Finish

Safe State

- **Safe state check: (P2,**

Need of P2<= Available

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

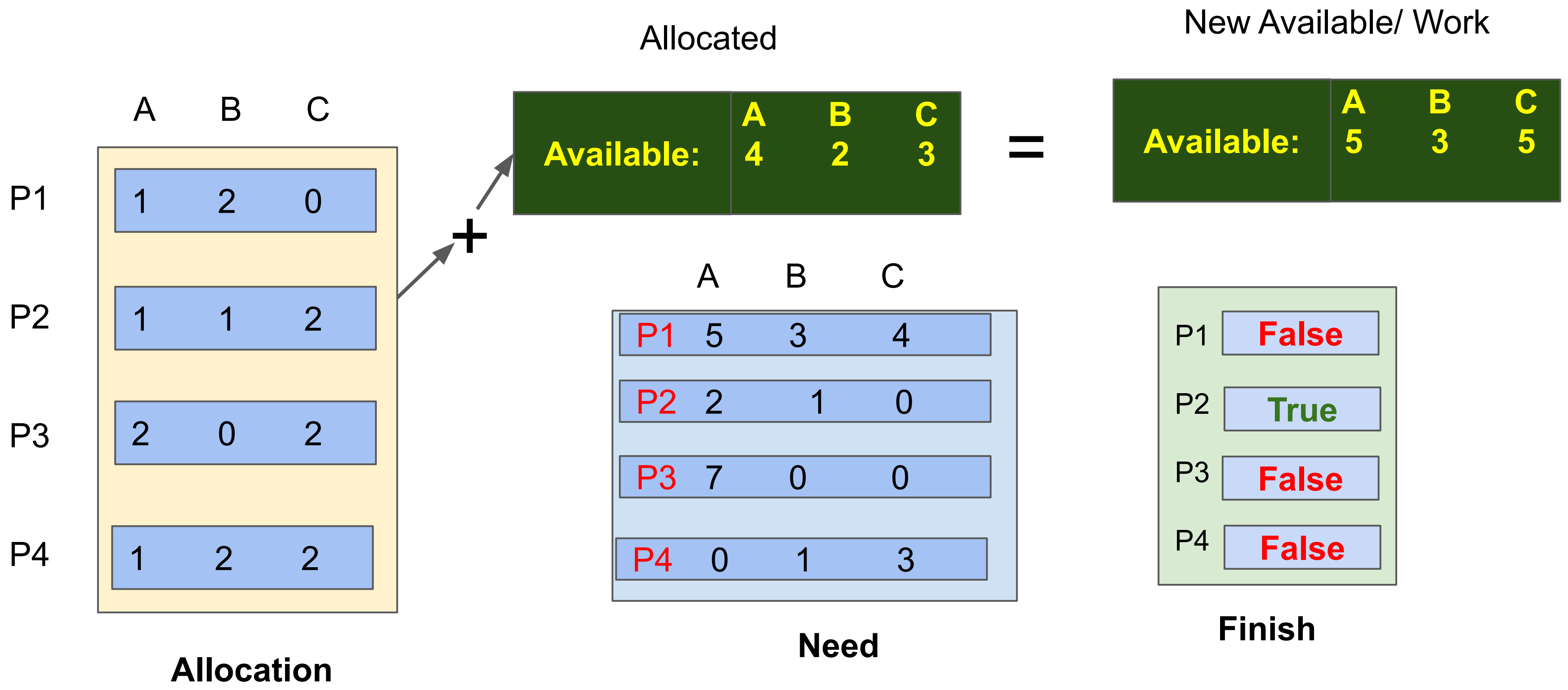
Need

<=

	A	B	C
Available:	4	2	3

Safe State

- **Safe state check: (P2,**



Safe State

- **Safe state check: (P2,_**

Need of P3 > Available

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

~~<=~~

	A	B	C
Available:	**5**	**3**	**5**

False
True
False
False

Finish

Safe State

- **Safe state check: (P2,P4,**

Need of P4<= Available

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

<=

	A	B	C
Available:	**5**	**3**	**5**

Safe State

- **Safe state check: (P2,P4**

Allocation

	A	B	C
P1	1	2	0
P2	1	1	2
P3	2	0	2
P4	1	2	2

+

Allocated

Available:	A	B	C
	5	3	5

=

New Available/ Work

Available:	A	B	C
	6	5	7

Need

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Finish

P1	False
P2	True
P3	False
P4	True

Safe State

- **Safe state check: (P2,P4,P1**

Need of P1 <= Available

Need

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

<=

	A	B	C
Available:	**6**	**5**	**7**

Safe State

- **Safe state check: (P2,P4,P1,**

Allocation

	A	B	C
P1	1	2	0
P2	1	1	2
P3	2	0	2
P4	1	2	2

Allocation

\+ Allocated

	A	B	C
Available:	6	5	7

=

New Available/ Work

	A	B	C
Available:	7	7	7

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

P1	True
P2	True
P3	False
P4	True

Finish

Safety Algorithm

- **Safe state check: (P2,P4,P1,P3)**

Need of P3 <= Available

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

<=

	A	B	C
Available:	7	7	7

Safe State

- **Safe state check: (P2,P4,P1,P3)**

Allocation

	A	B	C
P1	1	2	0
P2	1	1	2
P3	2	0	2
P4	1	2	2

+

Allocated

	A	B	C
Available:	**7**	**7**	**7**

=

New Available/ Work

	A	B	C
Available:	**9**	**7**	**9**

Need

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Finish

P1	**True**
P2	**True**
P3	**True**
P4	**True**

Safe State

- Since the finished matix is true for all process so the system has reached a safe state. Hence now the requested resource can be allocated to requesting process.

True
True
True
True

Finish

- So the safe allocation sequence is **P2,P4,P1,P3**

Resource Request Check

- **This algorithm checks if a new resource request is safe and grants resource only if it is safe.**

- For any new request R[i][j]
- Check **if(R[i][j] <= Need[i][j])**
    - **False: Can not grant request**
    - **True :** Check **if(R[i][j] <= Available[j])**
        - **False: Can not grant request**
        - **True: Available[j] := Available[j] - R[i][j];**
          Allocation[i][j] := Allocation[i][j] + R[i][j];
          Need[i][j] := Need[i][j] - R[i][j];

Resource Request Check

- **Initially suppose:**

Need

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Denotes number of resource of each type required by each process

Allocation

	A	B	C
P1	1	2	0
P2	1	1	2
P3	2	0	2
P4	1	2	2

Denotes number of resource of each type allocate to each process

	A	B	C
Available:	**4**	**2**	**3**

	A	B	C
P1 Requests:	**3**	**2**	**2**

Resource Request Check

- Check **if(R[i][j] <= Need[i][j])**

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

<=

P1 Requests:	A	B	C
	3	2	2

Need

Denotes number of resource of each type required by each process

Resource Request Check

- Check **if(R[i][j] <= Available[j])**

	A	B	C
Available:	4	2	3

<=

	A	B	C
P1 Requests:	3	2	2

Resource Request Check

- **Available[j] := Available[j] - R[i][j];**

	A	B	C
Available:	**1**	**0**	**1**

:=

	A	B	C
Available:	**4**	**2**	**3**

-

	A	B	C
P1 Requests:	**3**	**2**	**2**

- **Allocation[i][j] := Allocation[i][j] + R[i][j];**

	A	B	C
P1	4	4	2
P2	1	1	2
P3	2	0	2
P4	1	2	2

:=

	A	B	C
P1	1	2	0
P2	1	1	2
P3	2	0	2
P4	1	2	2

+

	A	B	C
P1 Requests:	**3**	**2**	**2**

Resource Request Check

- **Need[i][j] := Need[i][j] - R[i][j];**

	A	B	C
P1	2	1	2
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

:=

	A	B	C
P1	5	3	4
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

-

	A	B	C
P1 Requests:	**3**	**2**	**2**

Resource Request Check

- Finally we get the **new need, allocation and available** data.
- Now to determine if this new state is safe from deadlock we need to apply the **safely algorithm** on this new **need**, **allocation** and **available** data as before.
- If the new state is safe then we can immediately grant the request of P1, otherwise we have to reject it to make the state safe.

	A	B	C
P1	2	1	2
P2	2	1	0
P3	7	0	0
P4	0	1	3

Need

	A	B	C
P1	4	4	2
P2	1	1	2
P3	2	0	2
P4	1	2	2

Allocation

	A	B	C
Available:	1	0	1