

1 – Deadlock**List of Slides**

- 1 Deadlock
- 3 Deadlock
- 4 Resource Terminology
- 5 Conditions for Deadlock
- 6 Why Deadlock is an O/S level Problem
- 7 Some Useful Graph Theory
- 8 Resource Allocation graphs
- 11 Deadlock Management Strategies
- 12 Prevention
- 13 Detection/Recovery
- 14 Wait-For Graph Defined
- 15 Detection with Wait-For Graphs
- 16 Detection General Case

- 17 Reduction - An Example
- 18 Notes on Detection
- 19 Avoidance
- 20 Avoidance
- 22 Banker's Algorithm
- 23 Banker's Algorithm — Example 1
- 24 Banker's Algorithm – Example 1 part 2
- 25 Banker's Algorithm – Example 1 part 3
- 26 Banker's Algorithm — Example 2
- 27 Banker's Algorithm — Example 2, part2

2 – Deadlock

Deadlock is (informally) an indefinite postponement of a group of processes due to a conflict in their resource requirements, which could be granted in absence of the conflict.

An Example of Deadlock

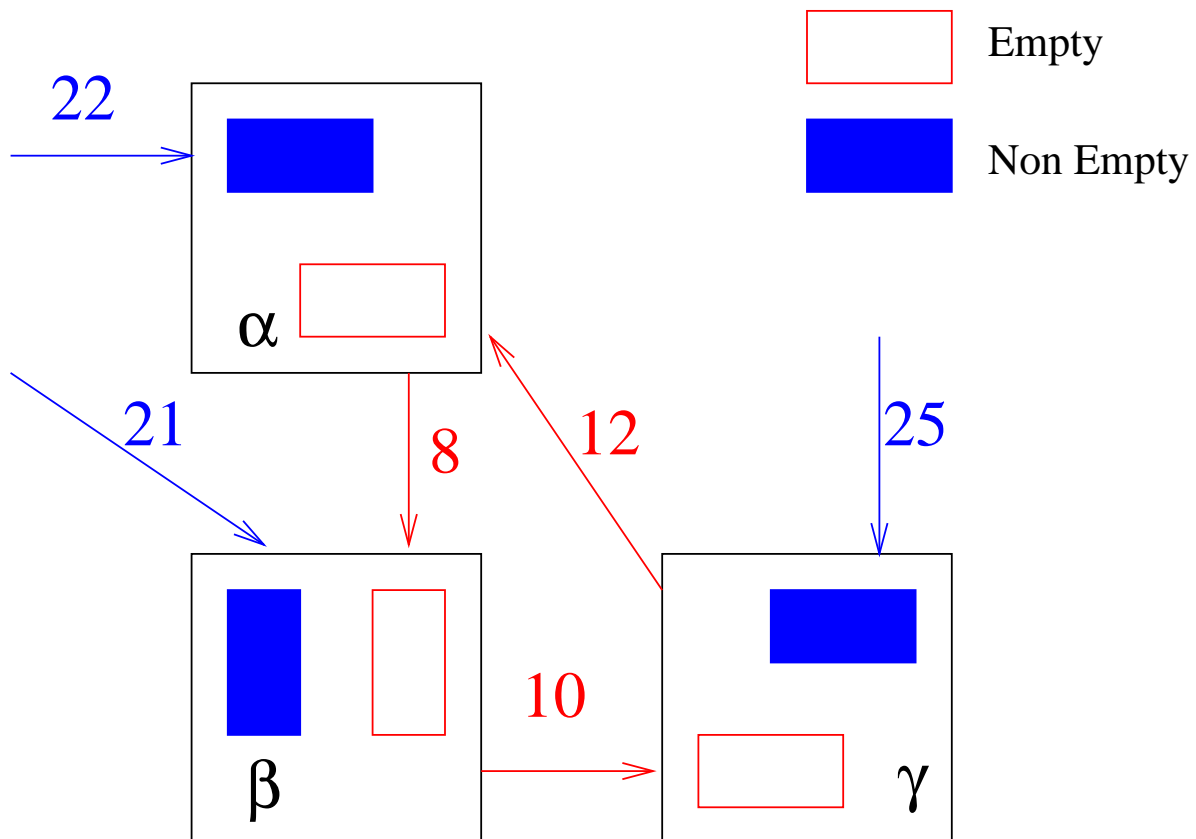


Figure 1: Deadlock

3 – Resource Terminology

Resources can be categorized as:

1. Reusable — Resources which have a fixed permanent total allocation, and are partitioned among processes.
2. Consumable — Resources can be created (produced) or destroyed (consumed), with processes req

Processes *request* a resources required for execution.

The operating system *grants* access to resources required by processes.

4 – Conditions for Deadlock

The necessary and sufficient conditions for deadlock are:

1. Mutual Exclusion — Necessary
2. Hold and Wait — Necessary
3. Nonpreemption of Resource Grants — Necessary
4. Circular Wait — Sufficient (can follow from the others)

5 – Why Deadlock is an O/S level Problem

Deadlock could be considered a programmer error, however multiple programs by different programmers can have unforeseen interactions.

6 – Some Useful Graph Theory

- A *directed graph* is a pair $G = (N, E)$ of nodes and edges.
- *Bipartite Graph* — Partition N into two sets such that all edges from nodes one set go into nodes in the other.
- *Sink* — A node with now outgoing edges.
- *Reachable Set of Node a* — all nodes on a path from a .
- *Knot* — A set of K nodes such that the reachable set of all nodes in K is K .

7 – Resource Allocation graphs

A system's current resource allocation state can be portrayed using a *resource allocation graph*.

The resource allocation graph is a bipartite graph containing:

1. Vertices, which are drawn as squares and circles:
 - (a) Processes — Drawn as Circles
 - (b) Resources — Drawn as boxes, multiple instances are shown as dots.
2. Edges, which are one of the following:
3. Request — From processes to resource (not obtained yet)
4. Grant — From a reusable resource to a process (indicates allocation)
5. Producer — From consumable resource to a process capable of producing it.

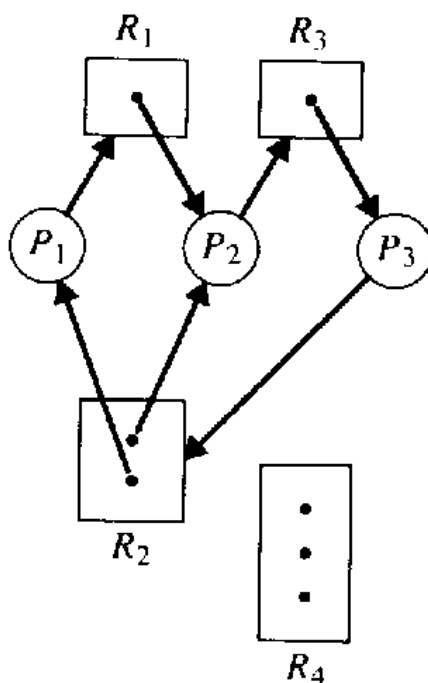


Figure 2: A Deadlocked Resource Allocation Graph

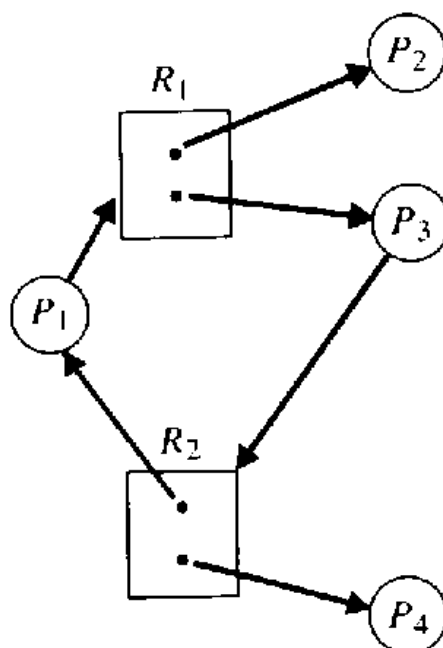


Figure 3: A Non Deadlocked Resource Allocation Graph

8 – Deadlock Management Strategies

Several approaches exist for deadlock management:

1. Ostrich — Ignore the problem and hope it never happens (surprisingly common in practice).
2. Prevention — Select a static policy to prevent one of the four conditions to hold.
3. Detection/Recovery — Identify deadlock when it occurs, and fix it then.
4. Avoidance — Detect requests which would lead to deadlock, and deny them as being unsafe.

9 – Prevention

Ensure that at least one of the following does not hold:

1. Mutual Exclusion — impractical for some resources.
2. Hold and wait — wait may induce starvation.
3. Non Preemption — impractical for many resource types.
4. Circular Wait — Impose a non cyclic order on resource request/grant ordering.
Sometimes difficult/impractical.

Prevention is only practical for certain resource types.

10 – Detection/Recovery

Detection involves running an algorithm periodically to test for deadlock in the system. Note that deadlock might be possible but not certain in a given configuration.

When deadlock is detected the operating system performs a *recovery* operation to fix the problem.

Recovery relies on:

1. Checkpointing — preserving process state
2. Rollback — state restoration to checkpointed state.

11 – Wait-For Graph Defined

The wait-for graph connects processes waiting on a resource to the process granted the resource *if there is a single instance of each resource type*.

Formally if process i is waiting for resource j which is granted to process k in the resource allocation graph then, the edge (i, k) appears in the wait-for graph.

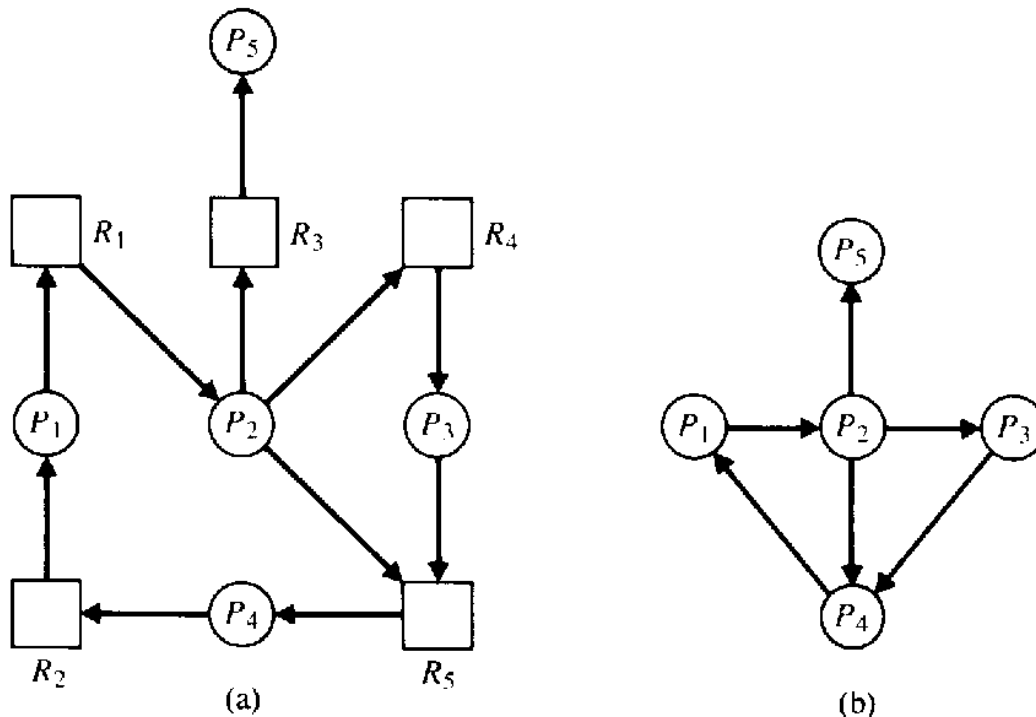


Figure 4: Wait-For Graph

12 – Detection with Wait-For Graphs

When using detection, if there is a single instance of each resource, it is possible to check for deadlock by searching for a knot in the corresponding wait-for graph.

If there are n processes, the test for cycles in the graph typically requires $O(n^2)$ operations.

13 – Detection General Case

Detection is done by *Graph Reduction*, for unblocked process P_i :

- For each reusable resource:
 - All of its request and grant edges can be removed.
 - Increase the available number of each reusable resource by the number of grant edges deleted.
- For each consumable resource:
 - Decrease the number of available units by the number of grant edges to P_i .
 - If P_i is a producer set the number of available units to ∞ .
 - Delete all edges between P_i and the resource.

14 – Reduction - An Example

Suppose that R_1 is a consumable resource and R_2 is a reusable resource.

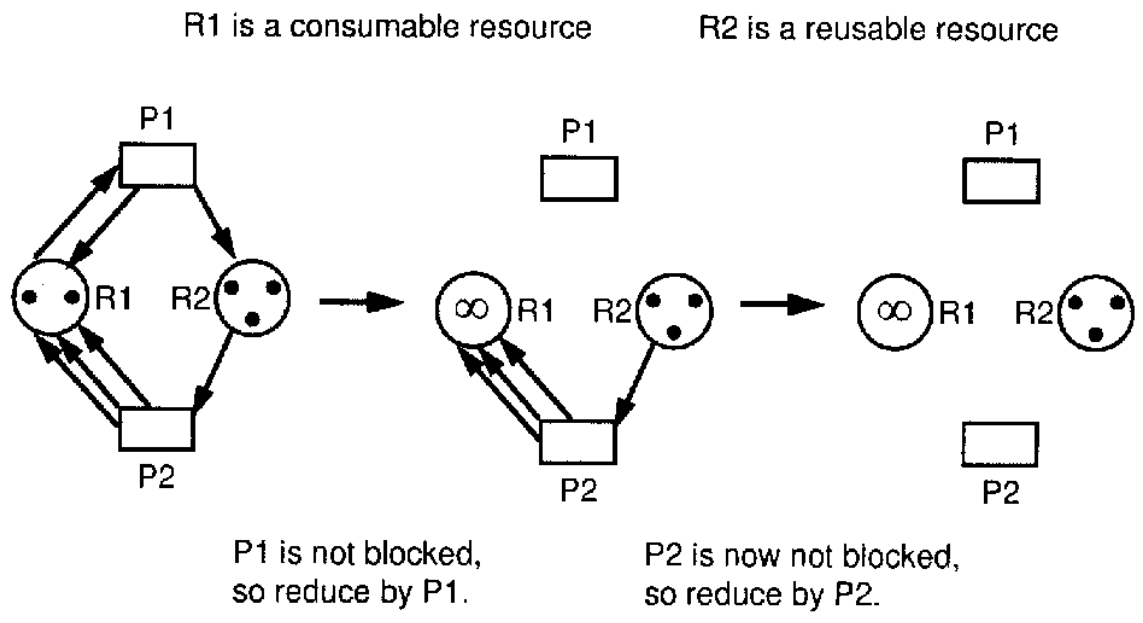


Figure 5: Reduce by P_1 which is unblocked, then P_2

15 – Notes on Detection

- Detection in the general case requires searching $n!$ reduction sequences for a reducible sequence.
- A general resource graph is *expedient* if all processes having requests are blocked.
- In a general resource graph a cycle is necessary for deadlock.
- For expedient resource graphs a knot is a sufficient condition for deadlock.
- Corollary: In an expedient graph if a process, P_i is not a sink and there is no path from P_i to a sink, then P_i is deadlocked.
- Converting from the general case to the special case is expensive.

Detection in the general case requires searching $n!$ reduction sequences

16 – Avoidance

The system can use additional information to determine how it should behave regarding granting a user's request.

If there is a reduction which leads to a potential deadlock, the system is in an *unsafe* state (with regards to deadlock). Avoidance prevents entry into unsafe states.

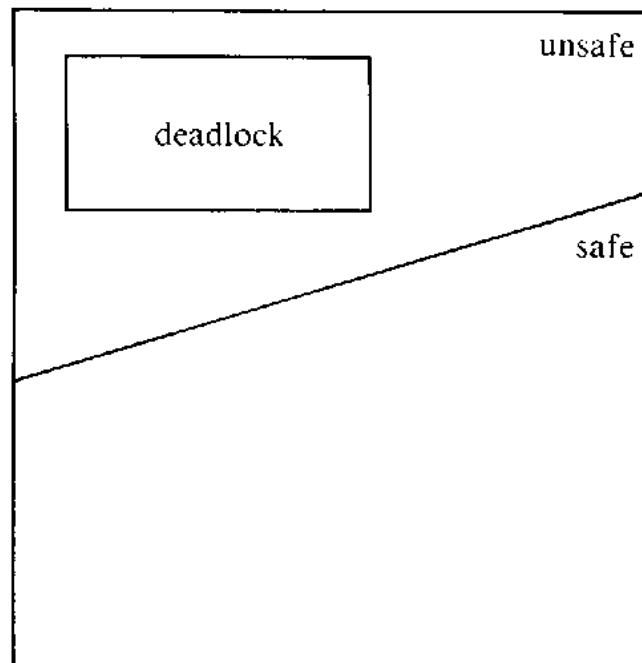


Figure 6: Safe, Unsafe and Deadlocked State Spaces

17 – Avoidance

The system state is given m resources and n processes:

$$1. R = \begin{bmatrix} R_1 \\ R_2 \\ \cdot \\ \cdot \\ \cdot \\ R_m \end{bmatrix} \quad \text{— a column vector of the total}$$

number each resource type.

$$2. AV = \begin{bmatrix} AV_1 \\ AV_2 \\ \cdot \\ \cdot \\ \cdot \\ AV_m \end{bmatrix} \quad \text{— a column vector of the}$$

unallocated number of each resource type.

$$3. C = \begin{bmatrix} C_{11} & \dots & C_{n1} \\ \cdot & \dots & \cdot \\ \cdot & \dots & \cdot \\ \cdot & \dots & \cdot \\ C_{1m} & \dots & C_{nm} \end{bmatrix} \text{ — a matrix of}$$

each processes requirements for each each resource type.

$$4. A = \begin{bmatrix} A_{11} & \dots & A_{n1} \\ \cdot & \dots & \cdot \\ \cdot & \dots & \cdot \\ \cdot & \dots & \cdot \\ A_{1m} & \dots & A_{nm} \end{bmatrix} \text{ — a matrix of}$$

each processes requirements for each each resource type.

18 – Banker's Algorithm

Dijkstra developed the banker's algorithm as a deadlock avoidance technique.

1. A *Safe state* has at least one reduction order in which all processes can complete.
2. Processes must tell the O/S the maximum allocation of each resource type which would be required when running that job.
3. Processes which would (if started) lead to an unsafe state are blocked until they are safe to run, i.e. to start process $k + 1$ if k processes are running requires that for all i , $1 \leq i \leq m$

$$R_i \geq C_{k+1,i} + \sum_{j=1}^k C_{j,i}$$

19 – Banker's Algorithm — Example 1

Suppose the system state is:

$$R = \begin{bmatrix} 9 \\ 3 \\ 6 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 6 & 3 & 4 \\ 2 & 1 & 1 & 2 \\ 2 & 3 & 4 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 1 & 5 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 2 \end{bmatrix} \quad AV = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

and Process 2 requests one unit each of R_1 and R_3 so that:

$$C = \begin{bmatrix} 3 & 6 & 3 & 4 \\ 2 & 1 & 1 & 2 \\ 2 & 3 & 4 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 1 & 6 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 2 & 1 & 2 \end{bmatrix} \quad AV = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Should the O/S grant this allocation (i.e. is this state safe)?

20 – Banker's Algorithm – Example 1 part 2

Complete process 2 by fulfilling its resource requirements (assigning it resource 3). The system becomes

$$C = \begin{bmatrix} 3 & 0 & 3 & 4 \\ 2 & 0 & 1 & 2 \\ 2 & 0 & 4 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix} \quad AV = \begin{bmatrix} 6 \\ 2 \\ 3 \end{bmatrix}$$

Then finish process 1:

$$C = \begin{bmatrix} 0 & 0 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 4 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix} \quad AV = \begin{bmatrix} 7 \\ 2 \\ 3 \end{bmatrix}$$

21 – Banker's Algorithm – Example 1 part 3

Finish process 3.

$$C = \begin{bmatrix} 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad AV = \begin{bmatrix} 9 \\ 3 \\ 4 \end{bmatrix}$$

Now process 4 can run, and all jobs in the system will complete normally.

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad AV = \begin{bmatrix} 9 \\ 3 \\ 6 \end{bmatrix}$$

So the state was safe.

22 – Banker's Algorithm — Example 2

Suppose the system state is:

$$C = \begin{bmatrix} 3 & 6 & 3 & 4 \\ 2 & 1 & 1 & 2 \\ 2 & 3 & 4 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 1 & 5 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 2 \end{bmatrix} \quad AV = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

and Process 1 requests one unit each of R_1 and R_3 so that:

$$C = \begin{bmatrix} 3 & 6 & 3 & 4 \\ 2 & 1 & 1 & 2 \\ 2 & 3 & 4 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 2 & 5 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 2 \end{bmatrix} \quad AV = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Should the O/S grant this allocation?

23 – Banker's Algorithm — Example 2, part2

No since the state is unsafe. Note that the system is not deadlocked, but that it merely has the potential for deadlock.

References