Deadlocks!

Resources

- A system consists of finite number of resources(like printers, tape drives, etc.) to be distributed among number of competing processes.
- Under the normal mode of operation, a process may utilize a resource in only the following sequence:

REQUEST → USE → RELEASE

Resources …

1. **Preemtable Resources**: Can be taken away from the process with no ill effects.
2. **Non-preemtable Resources**: Will cause the process to fail if taken away!

What is a deadlock?

A set of blocked processes is deadlocked if each holding a resource and waiting to acquire another resource held by another process in the set.

Example :

1. System has two disk drives.
2. P_1 and P_2 each hold a drive and each needs another one.

In general, deadlock involves nonpreemptable resources.

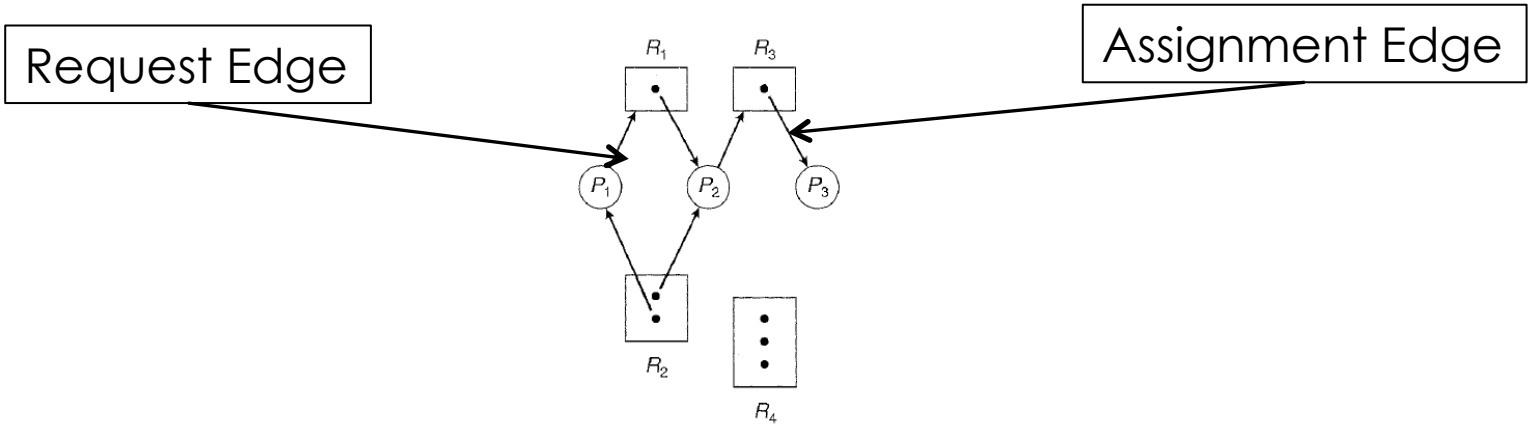
Necessary condition for Deadlock

1. Mutual Exclusion
2. Hold and Wait
3. No Preemeption
4. Circular Wait

Unlocking a deadlock is to answer 4 questions?

1. Can a resource be assigned to more than one process at once?
2. Can a process hold one resource and ask another?
3. Can resources be preempted?
4. Can circular wait exists?

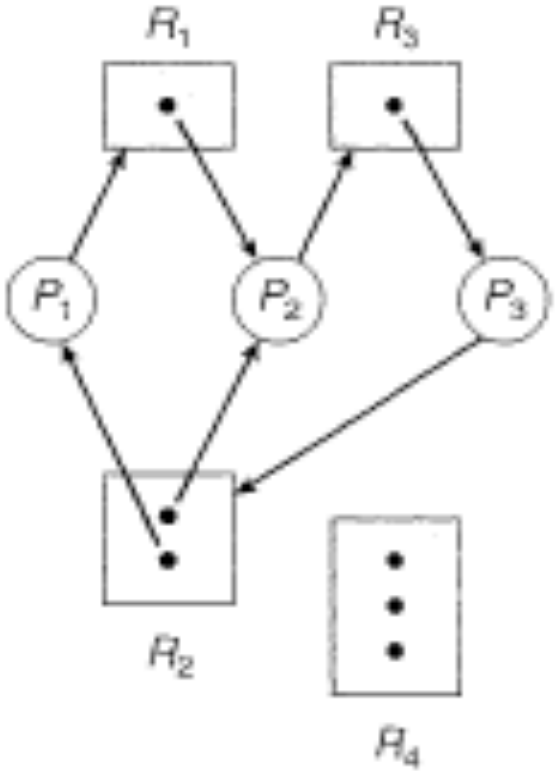
Resource Allocation Graph



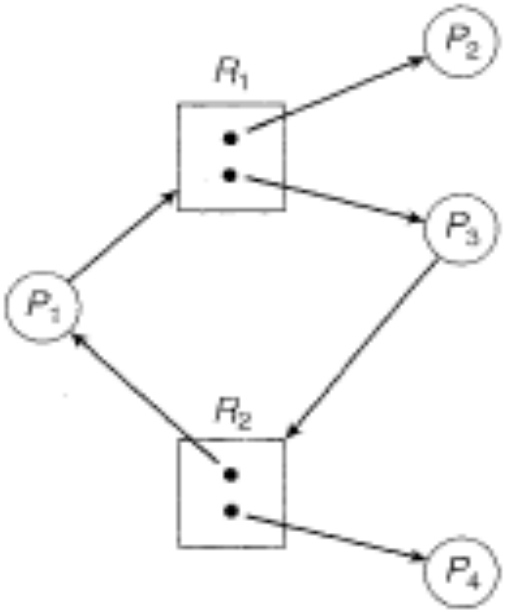
- Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1 .
- Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3
- Process P3 is holding an instance of R

Given the definition of a resource-allocation graph, it can be shown that, if the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock **may** exist.

Examples



Cycle and Deadlock



Cycle and No-Deadlock

Methods for handling deadlock

- Ignore the problem altogether
- Detection and Recovery
- Deadlock Avoidance
  - Careful Resource Allocation
  - Requires Prior Information of resource used by processes
- Deadlock Prevention
  - Negating one of the 4 necessary condition
  - By Constraining how request for resource can be made

Deadlock Prevention

Mutual Exclusion – Can't do anything about it!
Some Resources are intrinsically non shareable.

Deadlock Prevention…

Hold and Wait – To avoid this we need to make sure, process is not holding a resource while requesting for another.

- Request all resource before everything
- Request resource only if you hold none

Disadvantages

- Low Resource Utilization
- Starvation

Deadlock Prevention…

No Preemption–

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- Preempted resources are added to the list of resources for which the process is waiting.
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- If process is requesting another resource, if it is available then it is given to requesting process
- If it is held by another process which is waiting for another resource, we release it n give it to requesting process

Deadlock Prevention…

Circular Wait– Order Resource. Process can request resource only in increasing order of enumeration.

Process P holding R_i can request R_j only if $F(R_j) > F(R_i)$.

Deadlock Avoidance

- Needs additional information how resources will be requested by each process.
- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need

Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes!

A sequence of processes <P1, P2, ... , Pn> is a **safe sequence** for the current allocation state if, for each Pi, the resource requests that Pi can still make can be satisfied by the currently available resources plus the resources held by all Pj, with j < i.

Safe State Example:

	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2

Suppose processes P0, P1, and P2 share 12 magnetic tape drives. Currently 9 drives are held among the processes and 3 are available

Question: Is this system currently in a safe state?

Answer: Yes! Safe Sequence: <P1, P0, P2>

What if P_2 requests and is allocated one more tape drive?

How does safe state helps?

Key Ideas:

- Initially the system is in safe state
- Whenever a process requests for an available resource the allocation will only be made if the resulting state is safe.
- Otherwise, the process must have to wait.

Using Resource Allocation Graph for single instance of resource type!

Banker's Algorithm!

- RAL is not applicable for multiple instance of resource
- Bankers' algorithm - Multiple instances.
- Each process claims maximum resource needs *a priori*.
- When a process requests a resource it may have to wait.
- When a process gets all of its resources it must return them in a finite amount of time.

Banker's Algorithm – DS!

n= number of processes,
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].

Banker's Algorithm – Saftey Method

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   - Work = Available
   - Finish [i] = false for i = 0 to n- 1.
2. Find an i such
   - Finish [i] = false && $Need_i \leq Work$
   - 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.

Banker's Algorithm - Complete

Request = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j.

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:
   - Available = Available – Request;
   - $Allocation_i = Allocation_i + Request_i$;
   - $Need_i = Need_i – Request_i$
   - If safe ⇒ the resources are allocated to Pi.
   - If unsafe ⇒ Pi must wait, and the old resource-allocation state is restored

Banker's Algorithm - Example

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	

- What is the current state of system?
- Can request for (1,0,2) by P_1 be granted ?
- Can request for (3,3,0) by P_4 be granted ?
- Can request for (0,2,0) by P_0 be granted ?

Deadlock Detection

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	

- What is the current state of system?
- Can request for (1,0,2) by P_1 be granted ?
- Can request for (3,3,0) by P_4 be granted ?
- Can request for (0,2,0) by P_0 be granted ?

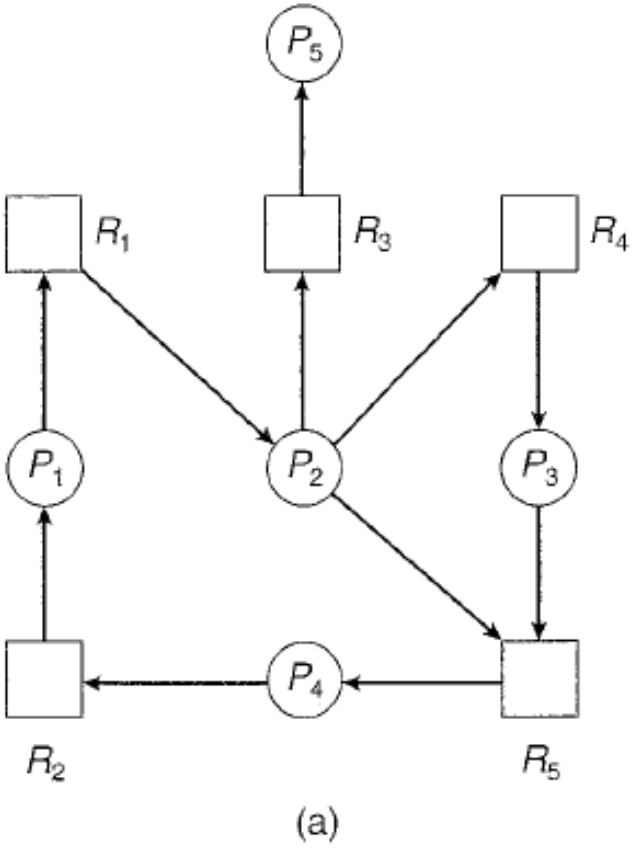
Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

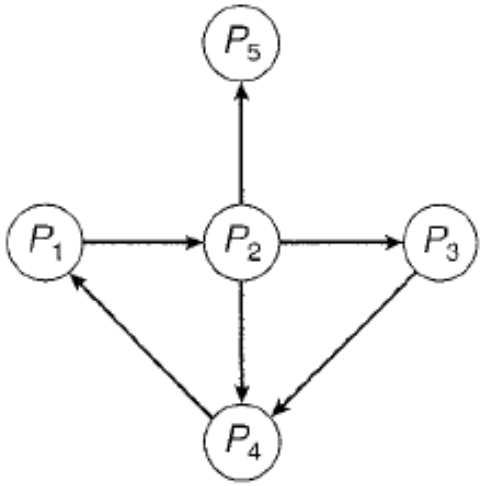
RAG and Wait – For Graph

- For Single Instance
- $P_i \rightarrow P_j$ (P_i is waiting for P_j to release a resource which P_i needs)
- $P_i \rightarrow P_j$ exist if and only if RAG contains 2 edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for some resource R_q
- A deadlock exists if and only if Wait – For Graph contains a cycle.
- To detect deadlocks, the system needs to maintain the wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph.

RAG and Wait – For Graph



(a)



(b)

CODING BLOCKS

What about Several Instance of Resource Type?

Several Instance of Resource - DS

- **Available**: A vector of length m indicates the number of available resources of each type.
- **Allocation**: An n x m matrix defines the number of resources of each type currently allocated to each process.
- **Request**: An n x m matrix indicates the current request of each process. If Request $[i_j] = k$, then process P_i is requesting k more instances of resource type. R_j.

Several Instance of Resource - Algo

1. Let Work and Finish be vectors of length m and n, respectively Initialize:
   - Work = Available
   - For i = 1,2, …, n, if $Allocation_i \neq 0$, then Finish[i] = false; otherwise, Finish[i] = true.
2. Find an index i such that both:
   - Finish[i] == false
   - $Request_i \leq Work$ 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] == false, for some i, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if Finish[i] == false, then P_i is deadlocked.

Example

Snapshot at time T_0:

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in *Finish*[*i*] = true for all *I*
- P_2 requests an additional instance of type C.
- State of System?

Detection Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back? -- One for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock

If Deadlock Then Recover!

Recovery – Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch

Recovery – Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.