Operating Systems [UCSC0503]

Deadlock

Prof. Tanaji B Patil

Department of Computer Science and Engineering

Contents

SYSTEM MODEL

- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$
  - CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $P_1, P_2, \ldots, P_n$ of waiting processes such that P_1 is waiting for a resource that is held by P_2, P_2 is waiting for a resource that is held by P_3, . . . , P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

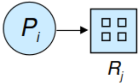
Resource-Allocation Graph I

- A set of vertices V and a set of edges E.
- V is partitioned into two types:
  - $P = P_1, P_2, \ldots, P_n$, the set consisting of all the processes in the system
  - $R = R_1, R_2, \ldots, R_m$, the set consisting of all resource types in the system
- **request edge** - directed edge $P_i \rightarrow R_j$
- **assignment edge** - directed edge $R_j \rightarrow P_i$

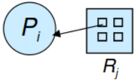
- Process

- Resource Type with 4 instances

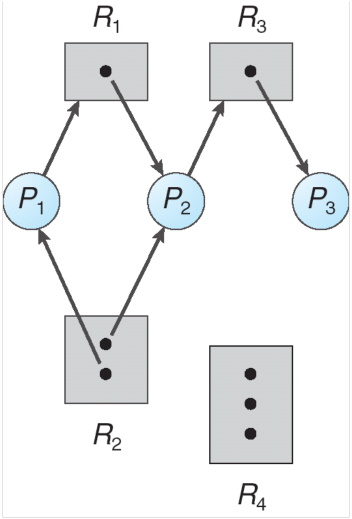
- P_i requests instance of R_j

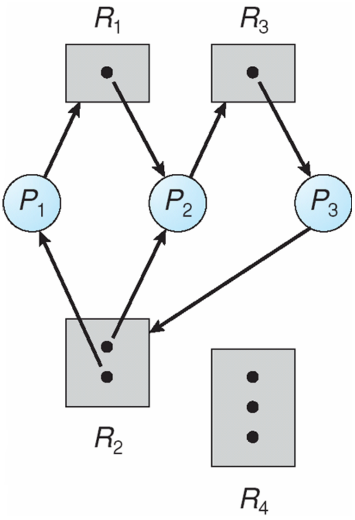


- P_i is holding an instance of R_j

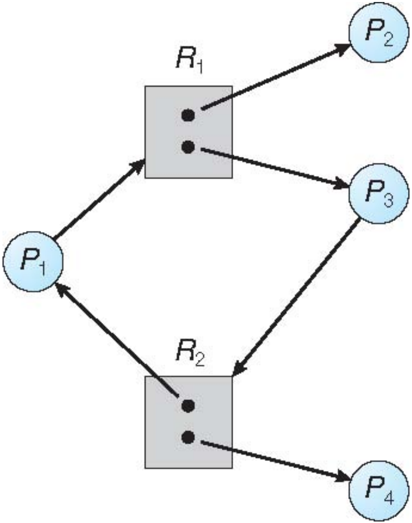


Example of a Resource Allocation Graph




R_1
R_3
P_1
P_2
P_3
R_2
R_4

Graph With Cycle but No Deadlock



Basic Facts

- If graph contains no cycles $\implies$ no deadlock
- If graph contains a cycle $\implies$
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Methods for Handling Deadlocks

- Ensure that the system will **never** enter a deadlock state:
  - Deadlock prevention
  - Deadlock avoidence
- Allow the system to enter a deadlock state and then recover
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

DEADLOCK PREVENTION

- **Mutual Exclusion** – not required for sharable resources (e.g., read-only files); must hold for non-sharable resources
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution
  - Low resource utilization; starvation possible
- **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
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance

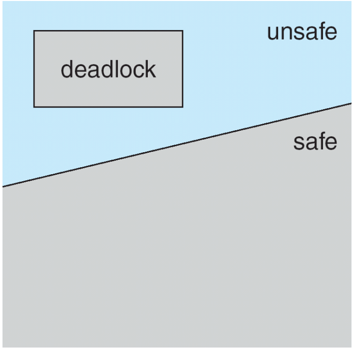
- Requires that the system has some additional a **priori** information available
- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state
- System is in **safe state** if there exists a sequence $< P_1, P_2, \ldots, P_n >$ of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources $+$ resources held by all the P_j, with $j < I$
- That is:
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, $P_i + 1$ can obtain its needed resources, and so on

Basic Facts

- If a system is in safe state $\Longrightarrow$ no deadlocks
- If a system is in unsafe state $\Longrightarrow$ possibility of deadlock
- Avoidance $\Longrightarrow$ ensure that a system will never enter an unsafe state.



Avoidance Algorithms

- Single instance of a resource type
  $\Longrightarrow$ Use a resource-allocation graph scheme

- Multiple instances of a resource type
  $\Longrightarrow$ Use the Banker's algorithm

RESOURCE-ALLOCATION GRAPH SCHEME

- **Claim edge** $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j; represented by a dashed line
- Claim edge converts to request edge when a process requests a resource
- Request edge converted to an assignment edge when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge
- Resources must be claimed a **priori** in the system

Example: Resource-Allocation Graph Scheme

The request $P_i \rightarrow R_j$ can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle

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

Let,
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**: $n \times m$ matrix.
  If $MAX[i, j] = k$, then process P_i may request at most k instances of resource type R_j

- **Allocation**: $n \times m$ matrix.
  If $ALLOCATION[i, j] = k$ then P_i is currently allocated k instances of R_j

- **Need**: $n \times m$ matrix.
  If $NEED[i, j] = k$, then P_i may need k more instances of R_j to complete its task

$$NEED[i, j] = MAX[i, j] - ALLOCATION[i, j]$$

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

   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

Resource-Request Algorithm for Process P_i

$Request_i =$ 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:

$$Available = Available - Request_i;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i;$$

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

- 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	

$Need = Max - Allocation$

	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.

- 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	

$Need = Max - Allocation$

	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: 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	

$Need = Max - Allocation$

	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 $Request \leq Available$ $((1,0,2) \leq (3,3,2)) \implies true$

	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	

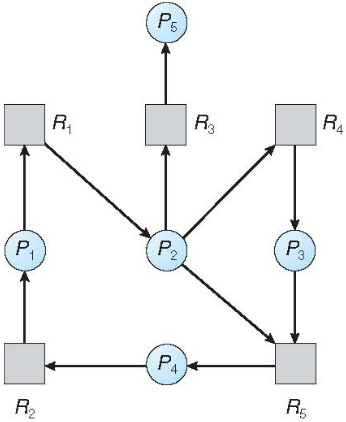
- Executing safety algorithm shows that sequence $< P_1, P_3, P_4, P_0, P_2 >$ satisfies safety requirement
- Can request for $(3,3,0)$ by P_4 be granted?
- Can request for $(0,2,0)$ by P_0 be granted?

Deadlock Detection

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

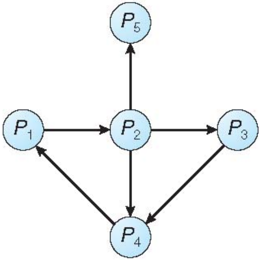
Single Instance of Each Resource Type

- Maintain **wait-for** graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph



(a)

Resource-Allocation Graph



(b)

Corresponding Wait-For Graph

- **Available**: A vector of length m indicates the number of available resources of each type
- **Allocation**: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process
- **Request**: An $n \times m$ 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.

1. Let **Work** and **Finish** be vectors of length m and n, respectively.
   Initialize:
   1. $Work = Available$
   2. For $i = 1, 2, \ldots, n$ if $Allocation_i \neq 0$, then $Finish[i] = false$ otherwise $Finish[i] = true$
2. Find an index i such that both:
   1. $Finish[i] = false$
   2. $Request_i \leq Work$

   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, then the system is in a deadlock state. Moreover, if $Finish[i] == false$, then P_i is deadlocked

Algorithm requires an order of $\mathcal{O}(m \times n^2)$ operations

Example: Detection Algorithm

- 5 processes P_0 through P_4
- 3 resource types: A (7 instances), B(2 instances), and C(6 instances)

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 $Finish[i] = true$ for all i

- 5 processes P_0 through P_4
- 3 resource types: A (7 instances), B(2 instances), and C(6 instances)

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 $Finish[i] = true$ for all i

Example: Detection Algorithm (Cont.)

- P_2 requests an additional instance of type C

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

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

Recovery from Deadlock: 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 from Deadlock: 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

Thank you . . .