CS240 Operating Systems, Communications and Concurrency

Deadlocks

Our model for the analysis of deadlocks consists of a multiprogramming environment where there is competition for a finite number of resources. A resource is any type of non-shareable object a thread/process needs to acquire to continue its execution.

The resources are grouped into different types. There may be a number of instances of each type available.

A queue of waiting processes/threads can be associated with each resource type. All the resources within a category are equivalent and a request for a resource in that category can be satisfied by any available instance.

CS240 Operating Systems, Communications and Concurrency

Tracking System Resource Usage

For all kernel managed resources, a system table could be used to record whether each resource instance is free or allocated and to which process it is assigned.

Applications may have their own programmer defined/user level synchronisation mechanisms and shared resources.

System could monitor deadlock of its own resources, but has no knowledge or control of application level resources shared among threads. For example, a Unix or Windows operating system could not check for deadlock within a Java Virtual Machine.

CS240 Operating Systems, Communications and Concurrency

Careful Programming

It is up to a programmer to ensure deadlock free execution of threads sharing application managed resources (and OS resources also) by following the correct usage of synchronisation mechanisms, (for example, not forgetting to release synchronisation mechanisms when done) and to have appropriate resource allocation strategies where deadlock may be possible.

Remember also that processes/threads may terminate due to an interrupt or exception in the code and held resources, like mutexes, may not be properly released.

CS240 Operating Systems, Communications and Concurrency

Resource Utilisation Protocol

A process utilises resources using the following protocol:-

Request - If instance of resource is not available, then requesting process must wait.

e.g. open(), malloc(), acquire(), create()

Use - Process uses instance of resource.

Release - Release resource when done.

e.g. close(), free(), release(), destroy()

CS240 Operating Systems, Communications and Concurrency

Definition of Deadlock

A set of processes is in a deadlock state when every process in the set is waiting on an event that can be caused only by another process in the set.

CS240 Operating Systems, Communications and Concurrency

Four Necessary Conditions for Deadlock

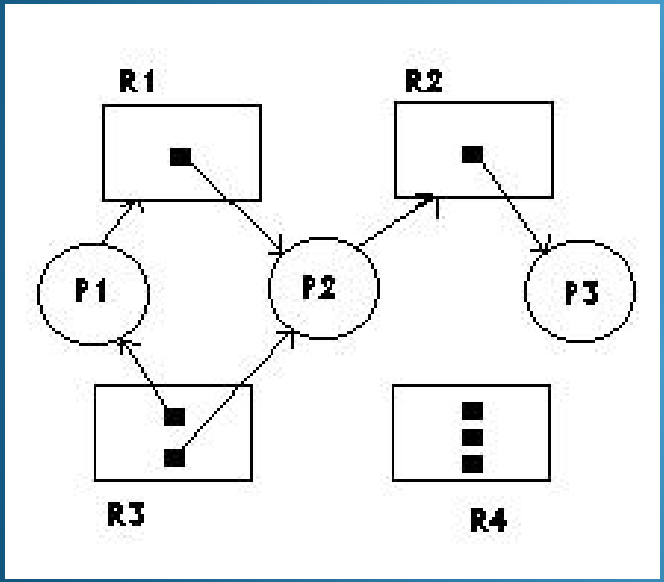
Mutual Exclusion - At least one resource must be held in a non-shareable mode.

Hold and Wait - A process must have at least one resource and be waiting for more.

No Preemption - Resources cannot be preempted. Resources are not released until process does so voluntarily.

Circular Wait - There must exist a set {P0,P1,P2,...,Pn} of waiting processes such that P0 waits for a resource held by P1, P1 waits for P2 ... and Pn waits for a resource held by P0.

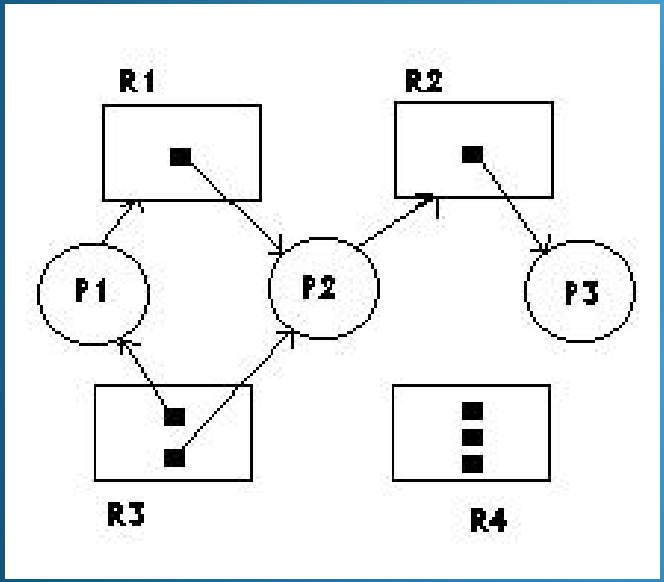
CS240 Operating Systems, Communications and Concurrency



The Resource Allocation Graph describes a snapshot of a system of processes and resources.

Resource types are represented by rectangles. Each resource type may have a number of instances represented by dots within the rectangle. An edge from a process to a resource type indicates that a process is waiting to acquire a resource of that type. An edge from a resource instance to a process indicates that the process has been allocated and holds that resource instance.

CS240 Operating Systems, Communications and Concurrency

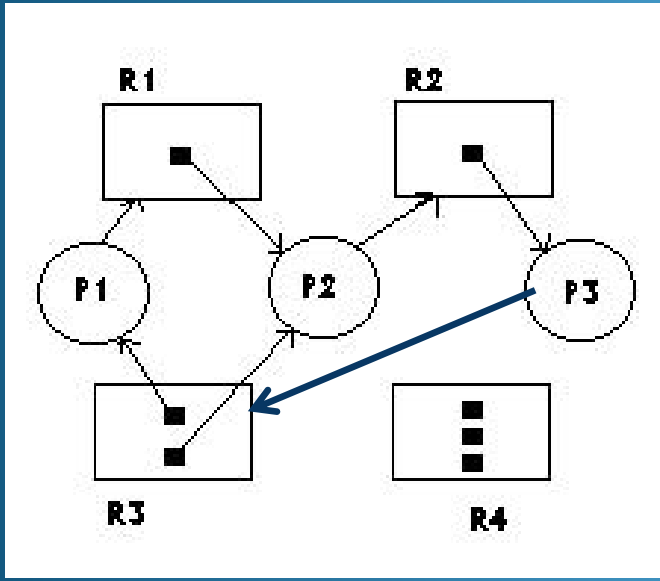


Identifying Deadlock

A deadlock can only exist if the graph contains a cycle, so if there are no cycles then the system is not deadlocked.

A cycle can be detected by a software algorithm which starts at a given node in the graph and follows all edges, marking visited nodes so that if a node is revisited during the traversal then a cycle exists.

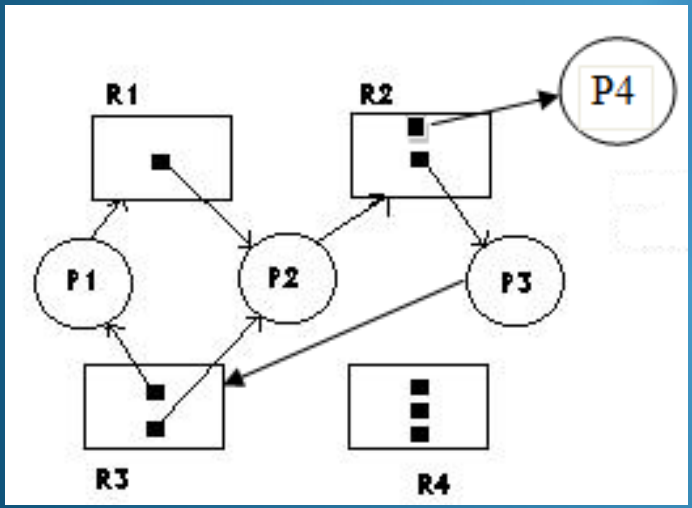
CS240 Operating Systems, Communications and Concurrency



Identifying Deadlock

The given graph has no cycles, however if P3 made a request for an instance of R3, then a deadlock would arise, involving all 3 processes in this situation, as shown on the left.

CS240 Operating Systems, Communications and Concurrency



Identifying Deadlock

A cycle is not a guarantee that a deadlock exists, but it is a necessary condition for the possibility of deadlock. A cycle could exist for example in the graph, but processes outside of the cyclic group may also hold resource instances that could break cycle when those resources are released.

In the graph, if P4 releases an instance of R2, this ends the wait of P2 which breaks the cycle.

CS240 Operating Systems, Communications and Concurrency

Methods for Handling Deadlock

1. Ensure system never enters deadlock.
    - 1.1 Deadlock Prevention
    - 1.2 Deadlock Avoidance

2. Allow system to enter deadlock and then recover.

3. Ignore the problem altogether (as in Unix).

CS240 Operating Systems, Communications and Concurrency

1. Ensure System never enters deadlock

1.1 *Deadlock Prevention* ensures at least one of the necessary conditions do not hold by generally constraining in some way how requests for resources are made.

The nature of some resources make them unshareable and so the mutual exclusion condition cannot be broken.

Also it may be difficult to preempt resource allocation unless this can be communicated in some way into the logic of the process.

CS240 Operating Systems, Communications and Concurrency

1. Ensure System never enters deadlock

1.1 *Deadlock Prevention* ensures at least one of the necessary conditions do not hold by generally constraining in some way how requests for resources are made.

For example, the Hold-and-Wait condition could be broken by requiring a process to request all resources at once, or by releasing existing resources before requesting more.

These methods can lead to starvation of processes that require a lot of resources or popular ones.

CS240 Operating Systems, Communications and Concurrency

1. Ensure System never enters deadlock

1.1 *Deadlock Prevention* ensures at least one of the necessary conditions do not hold by generally constraining in some way how requests for resources are made.

The Circular Wait condition could be broken by insisting processes request resources according to some predefined numeric sequence.

The consequence of these request restrictions generally is lower device utilisation and system throughput.

CS240 Operating Systems, Communications and Concurrency

1. Ensure System never enters deadlock

1.2 *Deadlock Avoidance* techniques supply the system with additional information about process resource needs over the lifetime of the process.

Rather than simply granting resources when they are available, the system instead grants them more cautiously i.e. only if there is a guaranteed sequence in which all processes can be executed to completion in the worst case scenario where they all might request their maximum resources at once.

CS240 Operating Systems, Communications and Concurrency

Deadlock Avoidance - The Bankers Algorithm

The Banker's Algorithm gets its name because it is a method that bankers could use to assure that when they lend out resources they will still be able to satisfy all their clients.

(A banker won't loan out a little money to start building a house unless they are assured that they will later be able to loan out the rest of the money to finish the house.)

CS240 Operating Systems, Communications and Concurrency

Deadlock Avoidance - The Bankers Algorithm

Each process declares the maximum number of each resource type that it may need over its lifetime. A system is "safe" if the system can allocate resources to each process in some order to allow each to complete.

The sequence <P1,P2,...,Pn> is safe if for each Pi, the resources that Pi can still request can be satisfied by currently available resources plus those held by all processes Pj where j<i.

If no such sequence exists the system is said to be unsafe, not necessarily deadlocked, but succeptible to deadlock. A deadlock could arise depending on the order that processes request remaining resources. The algorithm avoids this possibility.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Is the system safe?

Is there a sequence in which all processes can be executed if they all demanded their maximum resources simultaneously while holding existing allocations?

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2	1	2	2
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Is the system safe?
Could execute <P1> say.

Needs this many more of each type to execute and there are enough available to satisfy this.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	5 3 2
~~P1~~	~~2 0 0~~	~~3 2 2~~	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

P1 resources added to Available

Is the system safe?

P1 finished, now execute P3.

Sequence so far <P1, P3>.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	5 3 2
~~P1~~	~~2 0 0~~	~~3 2 2~~	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	

Is the system safe?
P1 finished, now execute P3.
Sequence so far <P1, P3>.

Needs this many more of each type to execute and there are enough available to satisfy this.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
~~P1~~	~~2 0 0~~	~~3 2 2~~	
P2	3 0 2	9 0 2	
~~P3~~	~~2 1 1~~	~~2 2 2~~	
P4	0 0 2	4 3 3	

P3 resources added to Available

Is the system safe?

P1 and P3 finished, now execute P4.

Sequence so far <P1, P3, P4>.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
~~P1~~	~~2 0 0~~	~~3 2 2~~	
P2	3 0 2	9 0 2	
~~P3~~	~~2 1 1~~	~~2 2 2~~	
P4	0 0 2	4 3 3	4 3 1

Needs this many more of each type to execute and there are enough available to satisfy this.

Is the system safe?

P1 and P3 finished, now execute P4.

Sequence so far <P1, P3, P4>.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 5
~~P1~~	~~2 0 0~~	~~3 2 2~~	
P2	3 0 2	9 0 2	
~~P3~~	~~2 1 1~~	~~2 2 2~~	
~~P4~~	~~0 0 2~~	~~4 3 3~~	

P4 resources added to Available

Is the system safe?

P1, P3 and P4 finished, now execute P2.

Sequence so far <P1, P3, P4, P2>.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 5
~~P1~~	~~2 0 0~~	~~3 2 2~~	
P2	3 0 2	9 0 2	6 0 0
~~P3~~	~~2 1 1~~	~~2 2 2~~	
~~P4~~	~~0 0 2~~	~~4 3 3~~	

Is the system safe?
P1, P3 and P4 finished, now execute P2.
Sequence so far <P1, P3, P4, P2>.

Needs this many more of each type to execute and there are enough available to satisfy this.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	10 4 7
~~P1~~	~~2 0 0~~	~~3 2 2~~	
~~P2~~	~~3 0 2~~	~~9 0 2~~	
~~P3~~	~~2 1 1~~	~~2 2 2~~	
~~P4~~	~~0 0 2~~	~~4 3 3~~	

P2 resources added to Available

Is the system safe?

P1, P3, P4 and P2 finished, now finally execute P0.

Sequence so far <P1, P3, P4, P2, P0>.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation	Max	Available
	A B C	A B C	A B C
~~P0~~	~~0 1 0~~	~~7 5 3~~	10 5 7
~~P1~~	~~2 0 0~~	~~3 2 2~~	
~~P2~~	~~3 0 2~~	~~9 0 2~~	
~~P3~~	~~2 1 1~~	~~2 2 2~~	P0 resources added to Available
~~P4~~	~~0 0 2~~	~~4 3 3~~	

Is the system safe?

P1, P3, P4 and P2 finished, now finally execute P0.

Sequence so far <P1, P3, P4, P2, P0>.

CS240 Operating Systems, Communications and Concurrency

Example

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Is the system safe?

Yes it is, the sequence <P1, P3, P4, P2, P0> could be followed without deadlock.

CS240 Operating Systems, Communications and Concurrency

2. Allow system to enter deadlock and then recover

Deadlock Detection requires an algorithm that examines the state of the system to determine if deadlock has occurred. An algorithm is then required to recover from the deadlock.

Deterministic algorithms for detecting graph cycles require O(|V|+|E|) time at best. E.g. Tarjan's Algorithm.

CS240 Operating Systems, Communications and Concurrency

2. Allow system to enter deadlock and then recover

How often should this algorithm be executed? Obviously there would be a performance hit caused by checking for deadlocks frequently.

Perhaps use a trigger to invoke deadlock detection such as when CPU utilisation drops below a threshold while unfinished processes remain in the system, or if a resource allocation request could not be granted due to resources being busy.

CS240 Operating Systems, Communications and Concurrency

2. Allow system to enter deadlock and then recover

How many processes will be affected by the deadlock recovery when it happens?

Deadlock resolution can be achieved by process termination or resource preemption.

(a) Aborting all processes

(b) Aborting one at a time until the deadlock is broken. How is the victim chosen? Priority? Time running? How many resources does it hold or need?

(c) Rollback of processes to earlier checkpointed states and replaying subsequent events from a log. This allows resources that were granted since the checkpoint to be preempted.