Deadlocks

Deadlocks

Deadlocks

A Computer system is made up of Resources.

A Resource has

Type (memory space, file, printer, . . .)

Instance (i.e. An example of a resource)

Example s

File A and File B are two instances of resource type File.
Printer LAB-1, Printer LAB-2, and Printer SC1507 are three instances of resource type printer

Deadlocks

A process may use a resource in only the following sequences:

- Request
  if request is not granted, then the requesting process waits on the resource.
- Use
  The process operates on the resource.
- Release
  The process releases the resource after use.

Deadlocks

A set of processes are in a deadlock state when every process in the set is waiting for an event that can take place only by another process in the set.

Example:

The set of process {P1, P2} are in deadlock if P1 is holding resource s1 and requesting resource s2. In the mean time, P2 is holding resource s2 and requesting resource s1.

Deadlocks

Let us have another example

The set of process {P1, P2, P3} are in deadlock if P1 is holding resource s1 and requesting resource s2 while P2 is holding resource s2 and requesting resource s3. In the mean time, P3 is holding resource s3 and requesting resource s1.

Deadlocks

OBSERVATIONS

- In the set {P1, P2, P3}, every process holds a resource and waiting on a resource held by next process.
  ∴ There is a circular wait in the set {P1, P2, P3}
- The resources are not sharable
  ∴ Use of these resources must be mutually exclusive.
- Any of the resources held by a process can not be preempted.
  ∴ The option of no-preemptive is true.
- It is clear that a process in the set is holding a resource and waiting for another resource.
  ∴ Every process is in "hold and wait" situation.

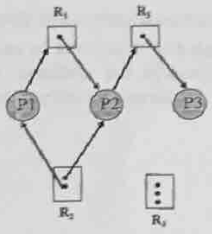
Deadlock characterization

Conclusion:

A deadlock situation can arise if the following four conditions hold simultaneously:

1. Circular wait
2. Mutual Exclusion (of resources)
3. No preemption (of resources)
4. Hold and wait

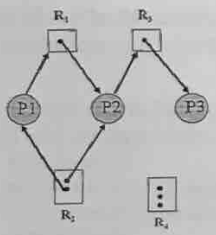
Resource Allocation Graph



To study deadlocks, resource allocation graph is needed

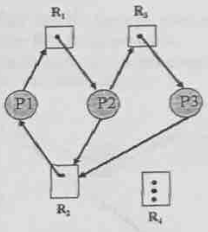
P= {P1, P2, P3}
R={R1, R2, R3, R4}
E={P1→R1, P2 →R3,
R1→P2, R2 →P2,
R2→P1, R3 →P3}

Resource Allocation Graph



If the graph has no cycles, then no process in the system is deadlocked.

Resource Allocation Graph



If the graph has a cycle and all the involved resources in the cycle have only one instance then there is a deadlock

P1→R1 →P2 →R2 → P1
P1→R1 →P2 →R3 → P3→R2 → P1

Resource Allocation Graph

Conclusions:

If the graph has no cycles, then no process in the system is deadlocked.

If the graph has a cycle and all the involved resources in the cycle have only one instance, then there is a deadlock

If the graph has a cycle and all the involved resources in the cycle do not have only one instance, then there may exist a deadlock

Methods for Handling Deadlocks

Deadlock prevention
Deadlock avoidance
Deadlock ignoring
Deadlock detection and recovery

Methods for Handling Deadlocks

Deadlock prevention

A set of methods ensuring that at least one of the four necessary conditions cannot hold.

Deadlock avoidance

Giving priori knowledge to OS regarding needs of processes to avoid deadlock.

Deadlock ignoring

Dead locks are ignored. Eventually the system comes to a holt, and then the system is re-booted.

Methods for Handling Deadlocks

Deadlock detection and recovery

A set of methods are used to detect deadlocks and another set of methods are used to recover from the deadlock.

Deadlock Prevention

A set of methods ensuring that at least one of the four necessary conditions cannot hold.

How?

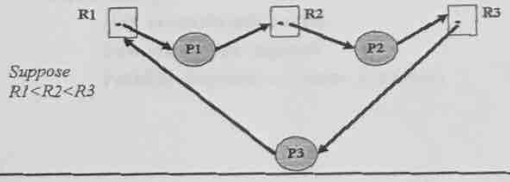
Let us look at the four necessary conditions again and analyze them.

1. Circular wait
2. Mutual Exclusion (of resources)
3. No preemption (of resources)
4. Hold and wait

Deadlock Prevention

Circular wait: assign a precedence to the resources' types and use the following protocol.

1- Each Process can request resources only in an increasing order of precedence.



Deadlock Prevention

Circular wait: assign a precedence to the resources' types and use the following protocol.

Each Process can request resources only in an increasing order of precedence.

Example:

P1 holds an instance of resource type Ri and requests an instance of Rj.

If Precedence Rj> Ri,

then request is granted.

else the instance of Ri must be released before an instance of Rj be given to P1.

Deadlock Prevention

Mutual Exclusion: we deal with non-sharable resources.

∴ we can not do anything about this condition

Deadlock Prevention

No Preemptive Condition: one of the following protocols does the job.

1- Process P1 currently holds r1, . . ., rn and requests rq. If rq can not be given to P1 immediately, then r1, . . ., rn are released.

2- P1 requests r1 that is hold by P2.
if P2 is waiting for additional resources,
then release r1
else P1 waits for r1.

Deadlock Prevention

Hold and wait: one of the following protocols does the job:

1- (Release before request)
A process requests resources only when the process has none. (i.e. request and release in a stepwise fashion)

2- (all or none)
Each process has to request and be allocated all of its requested resources before it begins execution.

Deadlock Prevention

Example:

P1 requests (a) tape drive, disk file, and (b) disk file and printer (for copying data from tape into a disk file, and print the file)

Protocol 1:
OS allocates tape drive and disk file to P1.
P1 releases tape drive and disk file before requesting disk file and printer.
disk file and printer are allocated to P1, if available.

Protocol 2:
OS allocates all three resources to P1, if available.
Otherwise, non of the resources are allocated to P1.

Deadlock Prevention

Advantage:
Preventing deadlock

Disadvantages:
Low resource utilization
Low system throughput
Possible starvation of some processes.

Deadlock Avoidance

Avoidance is possible by (a) having prior knowledge about the requests of processes and (b) granting the processes' requests in a way that system never enters a deadlock state.

How?

Deadlock Avoidance

Terminology

Resource allocation State
Safe Sequence
Safe State

Deadlock Avoidance

Terminology

At time t_0 we have:

1. A number of resources and each resource has a number of instances.
2. A number of processes that each one needs to use some or all of the resources.

At time t_i we want to know the state of resources (who is using them and how many resources are remained unused.
In other words,

What is the Resource allocation State at time t_i ?

Deadlock Avoidance

Terminology

Resource allocation State for time t_i is defined by:

(a) Maximum number of requested resources by each process at time t_0,
(b) The number of allocated resources to each process at time t_i , and
(c) The number of available resources at time t_i

Deadlock Avoidance

Example

We assume that 12 tape drives exist in the system at time t_0.
We also assume three processes of P0, P1, and P2 exist in the system

The Resource allocation State for time t_i :

(a) The maximum need for the processes are:
(b) The number of allocated resource at time t_i are:

	Max. need	# of currently allocated Resources
P0	10	5
P1	4	2
P2	9	3

(c) Number of available resources is 2.

Deadlock Avoidance

Safe Sequence

Given: a resource allocation state and
a sequence of processes $<P_1, \ldots, P_n>$
such that (P_j has a lower precedence than P_i if $j < i$ within the sequence)

The sequence of processes $<P_1, \ldots, P_n>$ is safe
If for each P_i, the resources that P_i requests can be provided for by:

- The available resources at the given state and
- The resources held by all processes with lower precedence than P_i

Deadlock Avoidance

Safe State

A system is in a safe state, if it encounters a safe sequence.

- **A safe state is free of deadlock.**
- **An unsafe state may create a deadlock.**

Deadlock Avoidance

To avoid a dead lock two categories of techniques are used:

a- Allocation of resources under the assumption that each resource has only one instance.
(Resource allocation graph algorithm)

b- Allocation of resources under the assumption that each resource has multiple instances.
(Banker's algorithm)

Resource Allocation Graph

Previously we showed resource allocation graph in which:

- represents a process
- represents a resource and instances of the resource
- Pi ⟶ Rj represents a request by process Pi for resource Rj
- Rj ⟶ Pi represents the allocation of resource Rj to process Pi

Resource Allocation Graph

we add one more type of edge to the graph called "claim edge" shown by:

Pi - - - - - - → Rj

Indicates a prior knowledge about Pi and Rj and it says that "Pi may request Rj sometime in future".

Resource Allocation Graph

When actually Pi request Rj and Rj is allocated to Pi then

Pi - - - - - → Rj

changes into

Pi ⟵——— Rj

After the Rj is released then again it changes into

Pi - - - - - → Rj

Resource Allocation Graph

Resource allocation graph algorithm

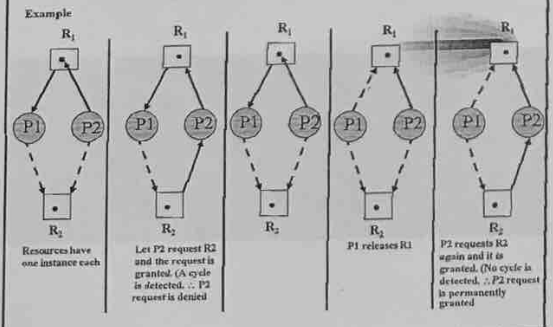
1- When Pi requests resource Rj, (i.e. Pi - - - - - → Rj), the request is granted (i.e. Rj ———→ Pi)

2- If the new edge creates a cycle in the graph, (considering all the edges including the claim edges)

Then the request is denied and Pi waits for the future changes in the graph that permits granting its request.

Resource Allocation Graph

Example



Deadlock Avoidance

Banker's Algorithms

- Safety Algorithm

  Determines whether a system is in a safe state.

- Resource-Request Algorithm

  Determines whether granting a resource-request put a system in a safe state.

Deadlock Avoidance

n the number of processes
(e.g. P = {P0, P1, P2, P3, P4}; n = 5)

m The number of resource types
(e.g. R = {A, B, C}; m = 3)

Is the sequence of <P1, P3, P4, P2, P0> safe?

Deadlock Avoidance

Notations (Cont'd)

Let us assume: Resources A, B, and C have 10, 5, and 7 instances, respectively.

Max[n X m] indicates the maximum demands of resources for each process

Allocation[nXm] indicates the number of resources of each type currently allocated to each process.

Need[nXm] indicates the remaining resource needs of each process. Need = Max - Allocation

Available[m] returns the number of available resources for every type.

Deadlock Avoidance

Notations (cont'd)

Let $(a_1, a_2, \ldots, a_k)$ and $(b_1, b_2, \ldots, b_k)$ be two vectors of X and Y, respectively.

If $b_i < a_i$ (for i = 1 to k), then Y < X.

If $b_j = a_j$ for some j value ($1 \le j < k$), and $b_j < a_j$ for the rest of j values, then $Y \le X$.

Example

If X = (1,3,7) and Y = (0, 3, 5), then $Y \le X$.

Safety Algorithm

1- work = available; //work is an array of m elements;
2- finish[n] = false //all elements;
3- Find an i such that both following conditions are true:
   a- finish[i] = false;
   b- Need[i,*] ≤ work //row i of need table;
   If no such i exist, then go to step 5;
4- work = work + allocation[i, *];
   finish[i] = true;
   Go to step 3;
5- If finish[i] = true (for i =1 to n)
   Then the system is in safe state;
6- End;

Resource-Request Algorithm

What does it do?

Resource-Request Algorithm

Given:
Three resource types of A, B, C,
Sequence of <P1, P3, P4, P2, P0> , and
A request from P2 for (1, 0, 2) instances of (A, B, C)

Does the request generate a deadlock?
The Resource-Request algorithm provides the answer

Resource-Request Algorithm

In a nutshell

Algorithm checks to make sure:

1. P2 request is not more than its need
2. P2 request is not larger than the number of available resources
3. If both checking is okay then, the **Need**, **Allocation**, and **Available** matrices are updated and safety algorithm is run
4. If the sequence is not safe, then deny the request by undoing all updates in Step 3 and P2 waits.

Resource-Request Algorithm

Let us formally present the algorithm

Resource-Request Algorithm

An array is needed to represent a process request of the resources.

$Request_i[m]$

This is an array of m elements. The j-th element of the array carries the number of instances of type j requested by process Pi.

For our example
$Request_i = (1,0,2)$ //request by P1

Resource-Request Algorithm

1- If ($request_i[*] > need[i,*]$)
   Then Error raised "Pi exceeds its maximum claim";
   Go to Step 5;
2- If ($request_i > available$) //Resources are not available.
   Then Pi waits; Go to Step 5;
3- // Allocate the requested resources to process Pi
   $available[i,*] = available[i,*] - request_i[*]$;
   $allocation[i,*] = allocation[i,*] + request_i[*]$;
   $Need[i,*] = Need[i,*] - request_i[*]$;
4- Invoke Safety (new state).
   If the state is not safe
   Then Pi waits; Undo step 3;
5- End;

Deadlock

So far we talked about :

- Deadlock ignoring
- Deadlock prevention
- Deadlock avoidance

The only deadlock handling method left to talk about is

- Deadlock detection and recovery

Deadlock Detection

If the system permits the occurrences of deadlocks, then we need to have algorithms for:

- Deadlock detection and
- Recovery from the deadlock

Deadlock Detection

Deadlock detection algorithms

- **Having only single instance for each resource type**
- **Having multiple instances for each resource type**

Deadlock detection algorithm: Having only single instance for each resource type

1-Generate a wait-for graph out of the resource-allocation graph

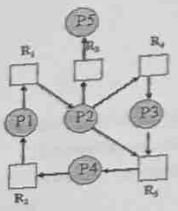
2- If the wait-for graph includes a cycle, then a deadlock has been detected.

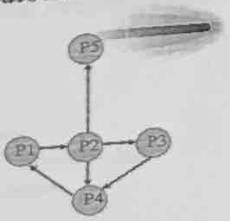
Wait-for Graph Creation

Assumption: Every resource has Only one instance

Given: A resource allocation graph
Objective: Generate a wait-for graph
1- Remove all the resource types from the resource-allocation graph by collapsing the edges of Pi⟶Rq⟶Pj into Pi⟶Pj;
2- End;

Deadlock Detection





A cycle exist in wait-for graph
∴ A deadlock is detected.

Deadlock detection algorithm: Having multiple instances for each resource type

Notations
n the number of processes and m The number of resource types
Let us assume A, B, and C have 10, 5, and 7 instances, respectively.

Allocation[nXm] indicates the number of resources of each type currently allocated to each process.

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Available[m] returns the number of available resources for every type

A	B	C
3	3	2

Request[nXm] indicates the current requests of each process.

	A	B	C
P0	5	4	3
P1	1	1	2
P2	2	3	0
P3	1	2	0
P4	2	1	4

Deadlock detection algorithm: Having multiple instances for each resource type

1- work[m] = available[m];
2- If allocation[i, *] ≠ 0 (for i = 0 to n-1)
   Then finish[i] = false;
   Else finish[i] = true;
3- Find an i such that both following conditions are true:
   a- finish[i] = false;
   b- Request[i, *] ≤ work[m] //row i of request table;
   If no such i exist, then go to step 5;
4- work[m] = work[m] + allocation[i, *];
   finish[i] = true;
   Go to step 3;
5- If finish[i] = false for some i (1 ≤ i ≤ n)
   Then the system is in a deadlock state.
   If finish[i] = false, Then Pi is deadlocked;
6- End;

Deadlock Recovery

Recovery is done by

1- Human Operator of the system manually or

2- The system automatically does

either

Process Termination,

or

Resource Preemption

Process Termination

1- Terminate all deadlocked processes

Expensive-- re-computing the aborted processes is costly.

2- Terminate one process at a time until deadlock is resolved.

Expensive-- invoking deadlock detection after each aborted process is costly.

Process Termination

How do you choose a process to be Terminated?

- Priority of the process.
- The amount of time that computer is into the process and the amount of remaining time for the process to be completed.
- How many resources has been used by the process.
- What type of resources has been used by the process. (knowing the type determines whether the resources are easy to regain or not.)
- How many more resources does the process need to be completed?
- Is the process interactive or batch?

Resource Preemption

Repeat until deadlock is resolved;

Preempt successively some resources and

Allocate them to other processes;

End;

Resource Preemption

Issues involved with resource preemption:

- Selecting a victim process to take away its resources
- Rollback the victim process
- Starvation