Lesson 16

Objectives

- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance

Methods for Handling Deadlocks

There are three approaches for handling deadlocks.

1. Ensure that the system will never enter a deadlock state by adopting some precautionary measures
2. Allow the system to enter a deadlock state and then recover
3. Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Precautionary Measures

There are two techniques upon employing any one of them would guarantee that system would not enter in a deadlock.

1. Deadlock Prevention
2. Deadlock Avoidance

1. Deadlock Prevention

In this technique we somehow make sure that at least one of the four necessary and sufficient conditions would not occur.

As we know that there is bi-conditional relationship between these conditions and the deadlock so if a single condition is missing system is prevented from deadlock.

Now let's examine these conditions one by one that how to get rid of them and what will be its possible cost.

1.1 Mutual Exclusion

Just imagine the dramatic situation that mutual exclusion was a necessary condition for the solution of critical section problem while in this case it is a candidate condition for the deadlock.

There are two types of resources that are sharable and non-sharable.

For sharable resources there is no need for Mutual Exclusion while for non-sharable resources this condition is unavoidable.

1.2 Hold and wait

In order to avoid the Hold and wait condition we have to do the following.

Solution Must guarantee that whenever a process requests a resource, it does not hold any other resources or if a process is already holding a resource it cannot request for another.

So it requires process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.

Problems

Starvation: The probability that all required resources of a process will be available is low, so process may be in an indefinite waiting state that is *starvation*.

Low resource utilization: Say for the instance that system is able to allocate all the required resources to the process simultaneously, then the process may be using anyone of them and rest of resources may not be in utilization this will cause poor resource utilization.

1.3 No Preemption

Solution

- If a process that is holding some resources requests another resource that can be immediately allocated to it, allocate it.
- 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.

Problem

Almost same problems may occur as it was in previous case.

1.4 Circular wait

Impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration. This will guarantee that circular wait wouldn't occur.

2. Deadlock Avoidance

Deadlock avoidance incorporates all a priori steps taken before execution of processes such that that sequence of processes may not produce a deadlock. It requires that the system has some additional a priori information available. Like,

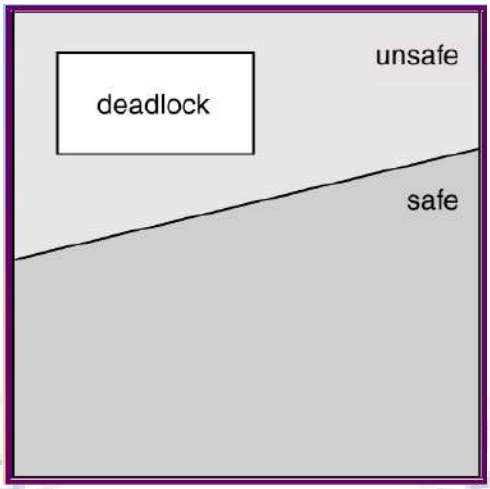
- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation state 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 safe sequence of all processes.
- Sequence <P1, P2, …, Pn> is safe if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j<I.
- If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished.
- When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.
- When Pi terminates, Pi+1 can obtain its needed resources, and so on.

Facts

- If system is in safe state then there is no possibility of deadlock
- If system is in unsafe state there may be a deadlock
- A safe sequence guarantees a safe state
- Deadlock avoidance is to make sure that system would never be in unsafe state



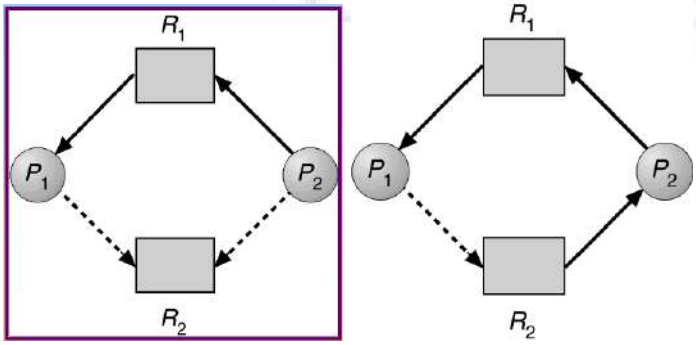
Safe and unsafe states

Avoidance Algorithm

a. RAG Algorithm (Single Instance per resource type)

Resource Allocation Graph is helpful in determining that when a resource/s should be allocated to which process. This can be done by introducing a new edge in resource allocation graph namely *Claim Edge*.

- Claim edge: Pi--->Rj indicated that process Pj may request resource Rj; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed a priori in the system
- System can a priory decide which process to give the resource. In following example if we allocate R2 to P2 then there is a deadlock while if we allocate to P1 there is no deadlock.



b. Banker's algorithm (Multiple instances per resource type)

- 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.

Example of Banker's Algorithm

- 5 processes P_0 through P_4; 3 resource types A (10 instances), B (5instances, 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	

- The content of the matrix. Need is defined to be 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.

- Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ *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 1	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?