Operating Systems (OS) Concepts - IT 308:
Lecture 14
Deadlock avoidance

**Batch:** B.Tech III year

**Instructor:** Rahul Muthu

DA-IICT
This is another way by which deadlocks do not occur. The *deadlock prevention* algorithms, discussed earlier stringently restrict the way in which processes may be allocated resources, often resulting in system under-utilisation.
Deadlock avoidance

- This is another way by which deadlocks do not occur. The *deadlock prevention* algorithms, discussed earlier stringently restrict the way in which processes may be allocated resources, often resulting in system under-utilisation.

- The *deadlock avoidance* algorithms rely on some degree of advance information on the demands on resources by various processes.
This is another way by which deadlocks do not occur. The *deadlock prevention* algorithms, discussed earlier stringently restrict the way in which processes may be allocated resources, often resulting in system under-utilisation.

The *deadlock avoidance* algorithms rely on some degree of advance information on the demands on resources by various processes.

To decide on any request, the system considers the resources currently available, the resources currently allocated to each process and the future requests and releases of each process. On the basis of this information, it decides whether the current request can be granted or to wait in order to avoid a possible future deadlock.
The simplest and most useful model in this scheme requires each process to declare the maximum number of each type of resource that it may need.
The simplest and most useful model in this scheme requires each process to declare the maximum number of each type of resource that it may need.

Using this information, the deadlock avoidance scheme ensures that there can never be a circular wait condition.
• The simplest and most useful model in this scheme requires each process to declare the maximum number of each type of resource that it may need.

• Using this information, the deadlock avoidance scheme ensures that there can never be a circular wait condition.

• The algorithm dynamically examines the resource-allocation state to control this process.
The simplest and most useful model in this scheme requires each process to declare the maximum number of each type of resource that it may need.

Using this information, the deadlock avoidance scheme ensures that there can never be a circular wait condition.

The algorithm dynamically examines the resource-allocation state to control this process.

The resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.
The system maintains the resource allocation related information using a *system table*.
The system maintains the resource allocation related information using a system table.

In this table it maintains a list of free resources and allocated resources.
The system maintains the resource allocation related information using a *system table*.

In this table it maintains a list of free resources and allocated resources.

Against each allocated resource, it also maintains the process number (PID) holding the resource.
A system is in a *safe state* if resources can be allotted to processes (up to their prespecified maximum requirement) in some order and still avoid a deadlock.
A system is in a *safe state* if resources can be allotted to processes (up to their prespecified maximum requirement) in some order and still avoid a deadlock.

Formally, a system is in a safe state only if there exists a *safe sequence*. 
A system is in a *safe state* if resources can be allotted to processes (up to their prespecified maximum requirement) in some order and still avoid a deadlock.

Formally, a system is in a safe state only if there exists a *safe sequence*.

A sequence of processes $\langle P_1, \ldots, P_n \rangle$ is a safe sequence for the current allocation state if, for each $P_i$, the resources that $P_i$ can still request can be satisfied by the currently available resources, plus the resources held by $\{P_1, \ldots, P_{i-1}\}$.
A system is in a *safe state* if resources can be allotted to processes (up to their prespecified maximum requirement) in some order and still avoid a deadlock.

Formally, a system is in a safe state only if there exists a *safe sequence*.

A sequence of processes $<P_1, \ldots, P_n>$ is a safe sequence for the current allocation state if, for each $P_i$, the resources that $P_i$ can still request can be satisfied by the currently available resources, plus the resources held by $\{P_1, \ldots, P_{i-1}\}$.

Thus, if the resources currently available are insufficient for $P_i$, it waits for the release of the resources by $P_1, \ldots, P_{i-1}$ after they complete.
$P_i$ then has the resources it needs to complete execution and terminate. At this point it also releases its resources to the common pool.
• $P_i$ then has the resources it needs to complete execution and terminate. At this point it also releases its resources to the common pool.

• When no such sequence exists, the system state is said to be *unsafe*. 
$P_i$ then has the resources it needs to complete execution and terminate. At this point it also releases its resources to the common pool.

When no such sequence exists, the system state is said to be unsafe.

A safe state is not a deadlock state.
- $P_i$ then has the resources it needs to complete execution and terminate. At this point it also releases its resources to the common pool.
- When no such sequence exists, the system state is said to be unsafe.
- A safe state is not a deadlock state.
- Conversely, a deadlock state is an unsafe state.
$P_i$ then has the resources it needs to complete execution and terminate. At this point it also releases its resources to the common pool.

- When no such sequence exists, the system state is said to be **unsafe**.
- A safe state is *not* a deadlock state.
- Conversely, a deadlock state is an unsafe state.
- However, not all unsafe states are necessarily deadlock states.
Below is a partial system table for a system with 12 instances of a tape drive.

Table: Multiple instances of a single resource

<table>
<thead>
<tr>
<th>Process</th>
<th>Maximum Needs</th>
<th>Current Needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>$P_1$</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

The question is, is this system in a safe state?