Operating Systems

Karim Lounis
Queen’s University, Canada

Winter’19, Jan-Apr 2019
Deadlocks, Deadlocks prevention, detection, and avoidance
Deadlocks

Definition

**Deadlock**: Is a system state in which processes or threads are waiting for some events (availability of some resources) that will never occur.

A set of processes $P = \{p_1, \ldots, p_n\}$ is deadlocked if: $\forall p_i \in P : p_i$ is waitn’ for an event that can only be caused by another process $p_j \neq i \in P$. 

[Image of a busy street scene]
Deadlocks

Definition

**Deadlock**: Is a system state in which processes or threads are waiting for some events (availability of some resources) that will never occur.

Resources:

- Resources allocation sequence: (1) A process requests a resources (thru a system call) (2) Gets the resources (3) Use the resources & (4) Releases the resources (thru a system call).
- Resources could be physical (e.g., I/O devices) or logical (e.g., data structures, messages, semaphores).
- A process cannot request a number of resources that exceeds the total number of available resources in the system.
Deadlock

Let $X$, $Y$, and $Z$, be three semaphores initialized to 0:

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acquire(X);</code></td>
<td><code>acquire(X);</code></td>
</tr>
<tr>
<td><code>Code 1;</code></td>
<td><code>Code 2;</code></td>
</tr>
<tr>
<td><code>release(X);</code></td>
<td><code>release(X);</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread 3</th>
<th>Thread 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acquire(Y);</code></td>
<td><code>acquire(Z);</code></td>
</tr>
<tr>
<td><code>Code 1;</code></td>
<td><code>Code 2;</code></td>
</tr>
<tr>
<td><code>release(Z);</code></td>
<td><code>release(Y);</code></td>
</tr>
</tbody>
</table>
Deadlocks

Definition

Deadlock: Is a system state in which processes or threads are waiting for some events (availability of some resources) that will never occur.

Deadlock is possible due to (Necessary Conditions):

1. **Mutual exclusion**: A resource (critical section) can only be allocated to one process at a time.

2. **No preemption**: A resource (critical section) cannot be taken away from any process during its critical section execution.

3. **Hold and Wait**: Once a resource (critical section) is taken, a process does not release it till it is done with it.

4. **Circular wait**: When each process $P_i$ is waiting for a resource held by process $P_{(i+1)[n]}$. 
There exists three main approaches to deal with deadlocks:

1. **Deadlock Ignorance**: It is the user who has to detect deadlocks and take actions (e.g., terminate processes or take back resources).
   
   Used by most operating systems e.g., Linux and Windows.

2. **Deadlock Avoidance and Prevention**: Ensure that the system will never enter into a deadlocked state:
   - **Deadlock Prevention**: This uses static rules for requesting resources. e.g., a process must request all needed resources at a time
   - **Deadlock Avoidance**: This uses dynamic rules. For each request, the OS determines whether a deadlock may arise or not before granting a given resource to a given process (e.g., Banker’s Algorithm).

3. **Deadlock Detection and Recovery**: Mostly used in DBMSs for transactions execution. Rollback if a deadlock is detected (i.e., two algorithms r needed, one to detect and one to recover).
Resource Allocation Graph

Definition

**RAG (Resource Allocation Graph).** is a directed graph where the vertices \( V = P \cup R \) represent the set of processes \( P = \{P_1, \ldots, P_n\} \) and system resources \( R = \{R_1, \ldots, R_m\} \), whereas the edges \( E \) express which process \( P_i \in P \) has requested or is holding a given resource \( R_j \in R \).

- An edge from process \( P_i \in P \) to a resource \( R_j \in R \); signifies that process \( P_i \in P \) has requested an instance of resource type \( R_j \in R \) and its currently waiting for it \( (P_i \rightarrow R_j \) is called request edge).
- An edge from a resources \( R_i \in R \) to a process \( P_j \in P \); signifies that process \( P_j \in P \) has been allocated an instance of resource type \( R_j \in R \) \( (R_i \rightarrow P_j \) is called assignment edge).
- Each process \( P_i \in P \) is depicted by a circle whereas resources \( R_i \in R \) are represented by rectangles.
Resource Allocation Graph

- $P_1$ requests $R_1$ and holds $R_2$.
- $P_2$ requests $R_2$ and holds $R_1$.

Deadlocks
We can use a RAG (Resource Allocation Graph) to identify whether a particular resource allocation sequence $S$ is deadlock-free or not.

If the RAG does not contain any closed path (cycle), then the sequence is **deadlock free**. Else **deadlock may or may not happen**.

- Could have multiple instances if a given resource.
- Could have processes that may release a given resource.
Resource Allocation Graph

Example. Resource Allocation Graph with cycle and deadlock.
Resource Allocation Graph

**Example.** Resource Allocation Graph with cycle but no deadlock.

![Resource Allocation Graph](image_url)
Resource Allocation Graph

**Extension.** A third type of edge exits, called *Claim Edge.*

[Diagram of resource allocation graph with nodes P1, P2, P3, R1, R2, and edges representing resource requests and claims.]
Deadlocks Prevention

In deadlock prevention systems, the system ensures that at least one of the four necessary conditions cannot hold.

1. **Mutual exclusion**: Must hold because of non sharable resources. Yet, for sharable resources, deadlock cannot happen.

2. **No preemption**: If a process has to wait, current resources are taken away (released). It is resumed if it gets all the resources it needs.

   Applied on resources whose state can be saved e.g., CPU and memory

3. **Hold and Wait**: Not allowing a process to hold resources and wait for other resources.

   Get all at a time or get without anything e.g., DVD-HDD-Printer

4. **Circular wait**: Impose a total ordering of all resource types while requesting e.g., An increasing order of enumeration.
Deadlocks Avoidance (Intro to Banker’s Algo)

Deadlock avoidance can be possible if the operating system holds a set of information:

- Total amount of each type of resource a.k.a., Resource Vector $R = \langle r_1, r_2, \ldots, r_n \rangle$.
- Total amount of each type of resource currently available a.k.a., Availability Vector $V = \langle v_1, v_2, \ldots, v_m \rangle$, $m \leq n$.
- Total amount of each type of resource needed by each process a.k.a., Claim Matrix $C(p \times n)$.
- Total amount of resources currently possessed by each process a.k.a., Allocation Matrix $A(p \times n)$.

Having this set of information, the operating system can operate the **Banker’s Algorithm** to determine whether a set of processes can be run till completion without deadlock (safe state) or not (unsafe state).
End.