Deadlock

Notice: The slides for this lecture have been largely based on those accompanying an earlier edition of the course text Operating Systems Concepts with Java, by Silberschatz, Galvin, and Gagne. Many, if not all, the illustrations contained in this presentation come from this source.
Concepts to discuss

- Deadlock
- Livelock
- Spinlock vs. Blocking
Deadlock: Bridge Crossing Example

- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.
Deadlock: Dining-Philosophers Example

Imagine all philosophers start out hungry and that they all pick up their left chopstick at the same time.

Assume that when a philosopher manages to get a chopstick, it is not released until a second chopstick is acquired and the philosopher has eaten his share.

**Question:** Why did deadlock happen? Try to enumerate all the conditions that have to be satisfied for deadlock to occur.

**Question:** How could be done to guarantee deadlock won't happen?
A System Model

• Resource types $R_1, R_2, \ldots, R_m$
  
  CPU cycles, memory space, I/O devices

• Each resource type $R_i$ has $W_i$ instances.

• Each process utilizes a resource as follows:
  – request
  – use
  – release
# Deadlock Characterization

Deadlock can arise if four conditions hold *simultaneously*:

- **Mutual exclusion**: only one process at a time can use a resource.
- **Hold and wait**: a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption**: a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait**: there exists a set \( \{P_0, P_1, \ldots, P_n\} \) of waiting processes such that \( P_0 \) is waiting for a resource that is held by \( P_1 \), \( P_1 \) is waiting for a resource that is held by \( P_2 \), \ldots, \( P_n \) is waiting for a resource that is held by \( P_{n-1} \), and \( P_n \) is waiting for a resource that is held by \( P_0 \).
Resource Allocation Graph

**Graph: G=(V,E)**

- The nodes in V can be of two types (partitions):
  - \( P = \{P_1, P_2, \ldots, P_n\} \), the set consisting of all the processes in the system.
  - \( R = \{R_1, R_2, \ldots, R_m\} \), the set consisting of all resource types in the system.
- request edge – directed edge \( P_i \rightarrow R_j \)
- assignment edge – directed edge \( R_j \rightarrow P_i \)
Resource Allocation Graph

- Process
- Resource Type with 4 instances
- $P_i$ requests instance of $R_j$
- $P_i$ is holding an instance of $R_j$
Example of a Resource Allocation Graph
Resource Allocation Graph With A Deadlock
Resource Allocation Graph With A Cycle But No Deadlock

Diagram showing the resource allocation graph with a cycle but no deadlock.
Basic Facts

- If graph contains no cycles $\implies$ no deadlock.

- If graph contains a cycle $\implies$
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.
Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.

- Allow the system to enter a deadlock state and then recover.

- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.
Deadlock Prevention

- **Mutual Exclusion** – not required for sharable resources; must hold for nonsharable resources.

- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources.
  - Require 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.
  - Low resource utilization; starvation possible.
Deadlock Prevention

Restrain the ways request can be made.

• **No Preemption** –
  – 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.

• **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.