What is Deadlock

**Formal definition:**
A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

Usually the event is release of a currently held resource.

**Informal:**
Every process is waiting for resource held by another process; none release until it gets what it is waiting for.

None of the processes can ...
- run
- release resources
- be awakened

Deadlock Representation

**Representing Deadlock**
Two common ways of representing deadlock:
- Vertices:
  - Threads (or processes) in system
  - Resources (anything of value, including locks and semaphores)
- Edges: Indicate thread is waiting for the other

**Wait-For Graph**

**Resource-Allocation Graph**
Conditions for Deadlock

Mutual exclusion
- Resource can not be shared
- Requests are delayed until resource is released

Hold-and-wait
- Thread holds one resource while waits for another

No preemption
- Previously granted resources cannot forcibly taken away

Circular wait
- Circular dependencies exist in “waits-for” or “resource-allocation” graphs

ALL four conditions MUST hold

Handling Deadlock

Ignore
- Deadlock only affects threads that cause it

Deadlock prevention
- Ensure deadlock does not happen
- Ensure at least one of 4 conditions does not occur

Deadlock avoidance
- Ensure deadlock does not happen
- Use information about resource requests to dynamically avoid unsafe situations

Deadlock detection and recovery
- Allow deadlocks, but detect when occur
- Recover and continue

Deadlock Prevention

No mutual exclusion --> Make resource sharable
- Example: Read-only files

No hold-and-wait --> Two possibilities
1. Only request resources when have none
   - Release resource before requesting next one
2. Atomically acquire all resources at once
   - E.g., single lock to protect all
   - Low resource utilization
   - Starvation
     - If need many resources, others might keep getting one of them

Deadlock Prevention (cont)

No “no preemption” --> Preempt resources
Example: A waiting for something held by B, then take resource away from B and give to A
- Only works for some resources (e.g., CPU and memory)
- Not possible if resource cannot be saved and restored
  - Can’t take away a lock without causing problems

No circular wait --> Impose ordering on resources
- Give all resources a ranking; must acquire highest ranked first
Deadlock Avoidance

Requires that the system has some additional \textit{a priori} information available.

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 and Unsafe States

System is in \textit{safe state} if there exists a safe sequence of all processes.

Avoid \textit{unsafe states} of processes holding resources

- Unsafe states might lead to deadlock if processes make certain future requests
- When process requests resource, only give if doesn't cause unsafe state
- Problem: Requires processes to specify all possible future resource demands

Dijkstra’s Banker’s Algorithm

Basic Facts

If a system is in \textit{safe state} => no deadlocks.

If a system is in \textit{unsafe state} => possibility of deadlock.

Avoidance => ensure that a system will never enter an unsafe state.

Banker’s Algorithm

Similar to reserving all resources at beginning, but more efficient

Each process must state maximum resource needs in advance
- but don’t actually acquire the resources

When thread later tries to acquire a resource, determine
- when it is \textit{safe} to satisfy the request
- block the thread if it is not safe
Deadlock Detection and Recovery

Detection
- Maintain wait-for graph of requests
- Run algorithm looking for cycles
  - When should algorithm be run?

Recovery: Terminate deadlock
- Reboot system (Abort all processes)
- Abort all deadlocked processes
- Abort one process in cycle

Challenges
- How to take resource away from process? Undo effects of process (e.g., removing money from account)
  - Must roll-back state to safe state (checkpoint memory of job)
- Could starve process if repeatedly abort it