System Model

* 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.
* System consists of resources type R1, R2, … , Rm
  + CPU cycles, memory space, IO devices, scanners, printers
* Each resource type has instances
* Each process utilizes a resource as follows
  + Request the resource
  + Use the resource
  + Release the resource

Deadlocks Characterization

* Must hold 4 conditions simultaneously

1. **Mutual exclusion**: one 1 process at a time can use a resource
2. **Hold and wait condition:** a process holding at least 1 resource is waiting to acquire additional resources held by other processes
   1. It’s now locked
3. **No preemption condition**: a resource can be released only voluntarily by the process holding it, after that process has completed its task
4. **Circular wait condition**
   1. There exists a set {P0, P1, … Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, … , P(n-1) is waiting for a resource that is held by Pn and Pn is waiting for a resource that is held by P0

Resource-Allocation Graph

* P is the set consisting of all processes
* R is the set consisting of all resources
* Request

Detect deadlock

* If graph contains no cycle -> no deadlock
* If graph contains a cycle
  + If 1 instance of one resource type, then deadlock (only 1 printer)
  + If more than 1 instance of a resource type, then no deadlock (2 printers)

Handling Deadlocks

* Ensure the system will never enter a deadlock state
  + Deadlock prevention
  + Deadlock avoidance
* Allow the system to enter a deadlock state and then recover
* Ignore the problem and present that deadlocks never occur, used by most operating systems

Deadlock prevention

* Mutual Exclusion: not possible
* Hold and Wait: must guarantee that whenever a process requests a resource, it does not hold any other resources -> not possible
* No preemption: not possible
* Circular wait: impose a total ordering of all resource types, and require process to request resource in increasing order

Deadlock avoidance

* Single instance
  + Use resource-allocation graph
* Multiple instances
  + Use the banker’s algorithm

Safe state

* If you can schedule at least 1 process can proceed with available resources, once it’s finished, you can gather the released - resources for another process