Deadlock

System Model

- **Computer systems** consist of **resources** (for example CPU, memory, and I/O devices).
- We also consider tools like **locks, and semaphores** to be **resources**.
- Each resource of type R_i has W_i instacnes.
- When a **process** utilizes a resource the following happens:
  1. The process **requests** the resource.
  2. The process **uses** the resource.
  3. The process **releases** the resource.

  – Processes can request as many resources as they need, it is not restricted. However, the number of resources requested cannot exceed the amount of resources available.
  – Processes may need to wait for resources to become available before their request is granted.
- The operating system uses a **system table** to record the status of **resources** (unallocated / allocated). If a resource is allocated, the process / thread allocating it is also recorded.

Deadlock

- A set of **threads** are said to be **in a deadlocked state** when **every thred** in the set is **waiting for an event** that can be caused only by **another thread in the set**.
- The following is an example of how deadlock can occur:
  1. Process one acquires a lock on resource 1.
  2. Process two acquires a lock on resource 2.
  3. Process wants to acquire a lock on resource two, but must block until it is available.
  4. Process two wants to acquire a lock on resource one, but must block until it is available.
  5. Both processes have entered deadlock.
- A set of **threads** are said to be **in a livelocked state** when **every thread** in the set is **waiting for an event** that can be caused only by **another thread in the set**. Livelock occurs when a the thread that the others are waiting for **continuously attempts an action that fails**.

Deadlock Conditions

- **Deadlock** can arise if **four conditions** occur at the same time:
  1. **Mutual exclusion**: Only one process at a time an use the resource.
  2. **Hold and wait**: A process holding at least one resource is waiting to acquire additional resources held by other processes.
  3. **No preemption**: A resource can be released only voluntarily by the process holding it.
  4. **Circular wait**: There exists a set of waiting processes such that P_0 is waiting for P_1, and P_1 is waiting for P_2 and ... and P_n is waiting for P_0.
- These conditions do **not guarantee deadlock** will occur, they can **potentially cause deadlock**.

Resource-Allocation Graphs

- A **resource-allocation graph** is a **directed graph**, $G = (V, E)$.
- V is partitioned into two sets:
  1. $P = \{P_1, P_2, ..., P_n\}$ consisting of all of the processes in the system.
  2. $R = \{R_1, R_2, ..., R_m\}$ consisting of all of the resource types in the system.
- E consists of two types of edges:
  1. A **request edge** is a directed edge of the form $P_i \to R_j$.
  2. An **assignment edge** is a directed edge of the form $R_j \to P_i$.
- If G has a **cycle**, **deadlock** may exist.
- If G has **no cycles**, **deadlock** does not exist.

Methods for Handling Deadlock

- One way to handle deadlocks is to ensure the system will never enter a deadlocked state (prevention / avoidence).
- Another way to handle deadlocks is to allow the system to enter a deadlocked state, and then recover.
- The last way is to ignore the problem, and pretend that deadlocks never occur in the system (this option is used by most systems).