**DEADLOCKS**

**Introduction**

* In multiprogramming system, **processes** **compete** over **limited resources**.
* If **no** resource is available, then processes enter **waiting state**.
* Sometimes a deadlock is created when processes are **unable to use a resource** because a **more competitive process** always uses it when it is made available.

**System Model**

* **Step 1:** **All** **processes** will **request** for the resource.
* **Step 2:** If it is available, then the **most** **competitive process** will use it.
* **Step 3:** After using it, the process **must not** use this resource until **rest of the processes** which requested for it have used it as well.

**Conditions for Deadlock**

* Mutual exclusion
* **Hold and wait:** A process using resource(s) requests for **even more**, busy resources.
* No pre-emption
* **Circular wait:** All processes waiting for another process, making a **circular structure**.

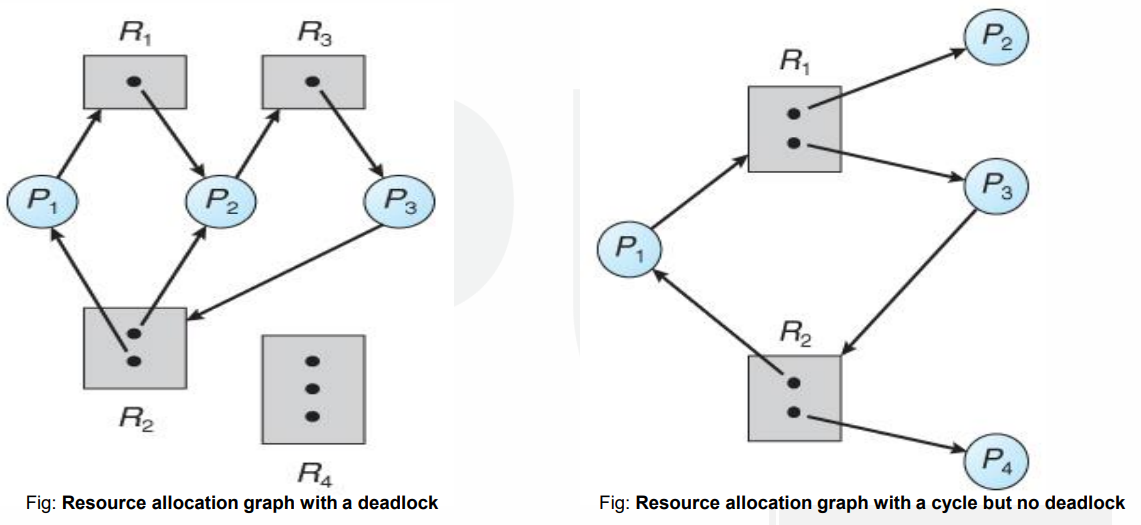
**Resource Allocation Graph**

* Used for **understanding** deadlocks.
* Its components are:
  + **Resources set**
  + **Process set**
  + **Resource instance**
  + **Request edges**
  + **Assignment edges**

Components description:-

* **Resources set** = ***{R1, R2, R3, … , Rn}***
* **Process set** = ***\*Now you know\****
* **Resource instance:** **Dots** representing **many resources** of a particular category.
* **Request edges:** **Edges** representing which **process** **requested** which **resource**.
* **Assignment edges:** Same as **request edge**, but **reversed**.

Graph conditions for deadlock:-



***\*Note: These are directed graph\****

* **Cyclic structure** is formed by **directed edges**.
* Each resource category contains **one** instance.
* But if **multiple instances** are there rather, then it shows there **might be** a deadlock.

**Deadlock Handling Methods**

* **Prevention:** **At least one** deadlock condition must **not** be satisfied.
* **Avoidance: Ignoring data** that may entertain deadlocks.
* **Detection and recovery:** **Recovering** a deadlock when it is detected.
* **Ignorance:** We **ignore** the problem as if it does **not** exist.

**Deadlock Prevention**

By removing mutual exclusion:-

* If a resource is made **concurrently sharable**, then deadlock will **not** **occur**.
* But some **hardware resources** like flash drives, disks and printers **aren’t sharable**.

By removing hold & wait:-

* **Conservative approach:** Process is allowed to execute **only** when it **naturally holds** all the resources.
* **Do not hold:** A process will get desired resources; but when making more request, it must release **previously held** resources.
* **Wait timeouts:** A process gets limited time to hold the resources with it.

By no pre-emption:-

* **Forceful pre-emption:** **High priority** processes are allowed to **pre-empt** over resources.

By circular wait:-

* Every process is assigned a **natural number**, for being **executed serially**.
* This can be either in **increasing** or **decreasing** order.

**Deadlock Avoidance**

* When scheduler **suspects** a process for creating a **future deadlock**, it **denies** resource access request to it.
* **Resource allocation state:** Current **condition of resources** in our system.
* **Example:** Like how many resources are allocated, how many are not etc.
* **Safe state:** A system state in which the **resource requirements** of all the processes are **met**, without creating a deadlock.
* Also, it is considered **safer** if the resources allocated by processes are **freed** **after execution**, so that upcoming processes can use them **smoothly**.

**Banker’s Algorithm**

* Whenever a process **starts executing**, it tells how much resources it will use **at max**.
* When the scheduler receives request for **resource using**, scheduler checks whether granting it would leave the system in **safe state**.
* If not, then scheduler **waits** for the **right time** to do so.
* Also known as **resource request algorithm**.

Code structure:-

**m = Number of resource categories**

**n = Number of processes**

**Available[m] = Number of resources in each category**

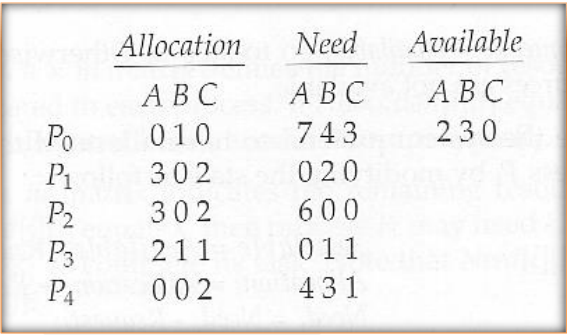
**Max[n][m] = Maximum number of resources a process is demanding**

**Allocation[n][m] = Number of resources allocated to a process**

**Need[n][m] = Number of resources a process is short of**

Safety algorithm:-

* An algorithm that tells whether the state is **safe or not**.
* Let **Work** and **Finish** be vector arrays of length **m** and **n**.
* **Work** is **available resources** for each process.
* **Finish** contains **completion status** of each process, in **Boolean**.
* By default, **Work** is **available** & **Finish** is **false** for all elements.
* ***\*Rest of it is code based, better to not discuss in theory\****

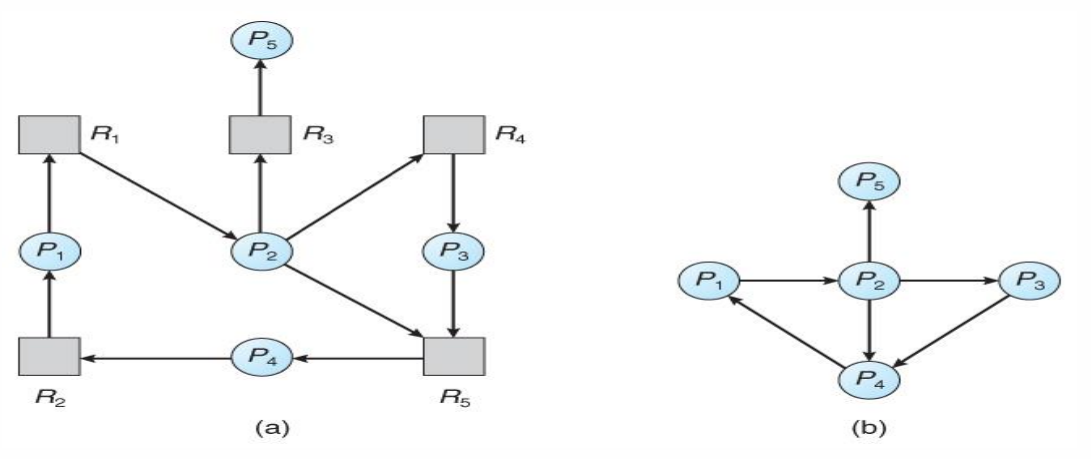


**Deadlock Detection**

* It is done when deadlock have **somehow** **occurred**, and now we want to **detect** it.
* Deadlock detection works by checking for deadlock by **scanning whole system**.
* But along with this technique, comes risk of **losing unsaved data**.

Wait-for graph:-

* If only **one instance** of each resource is there, then we **don’t** need to represent resources on graph.
* We can connect the lines **removing the boxes**.
* This is called **wait-for graph**.
* Here also a cycle represents **deadlock**.



* There are **two differences** between this **detection technique** & **banker’s algorithm** in code.
* The **frequency of scan** for detecting deadlock depends on **how** the detection algorithm is programmed.
* The detection algorithm is programmed as per the **expected frequency** of deadlock occurrence.
* And also programmed as per the **required seriousness** in the situation of deadlock.
* One common way this algorithm is programmed is by scanning for deadlock immediately after any **resource is allocated** to a process.
* But this approach increases the **overhead** on the system.
* It is advised to use deadlock when some strange performance is noticed in CPU.

**Deadlock Recovery**

Two methods:-

* Process termination
* Resource pre-emption

**Process Termination**

* We **kill** the **processes** involved in it.
* Methods to terminate:
  + **Method 1:** **Abort** all the processes **involved** in deadlock.
  + **Method 2:** **Abort** processes in deadlock **one-by-one**; until deadlock is removed. And then scanning is done for confirmation.

**Resource Pre-emption**

* We **transfer** some resources from **deadlocked processes** to **other processes**.
* Issues raised:
  + **Issue 1:** We have to **figure out** which **resources** & **processes** will be pre-empted.
  + **Issue 2:** We also have to **figure out** what to do with the pre-empted processes.
  + **Issue 3:** Pre-empting same process every time may lead to **starvation**. It can be solved by pre-empting each process **limitedly**.
* **Starvation:** A process never being able to complete its task.