SEMAPHORES

Semaphore is a very popular tool used for process synchronization.

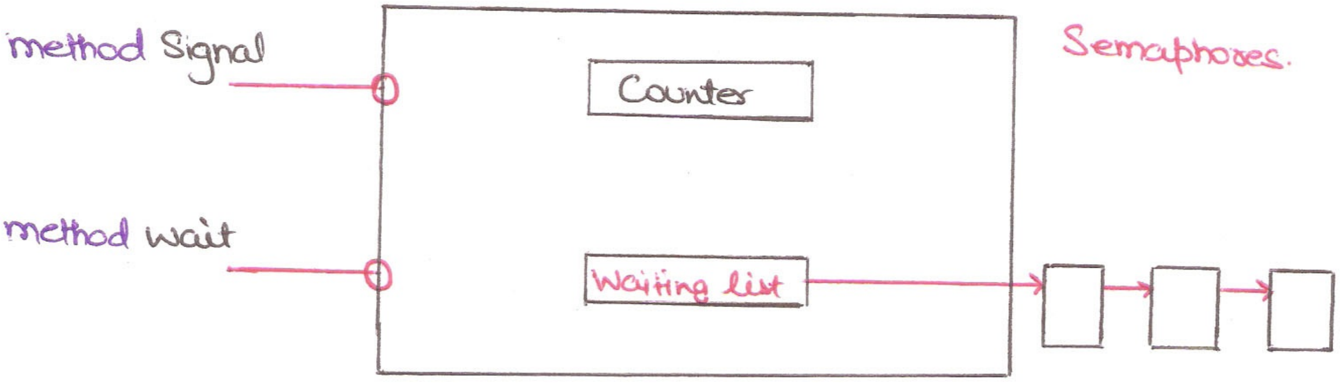
Semaphore is used to protect any resources such as global shared memory that needs to be accessed and updated by many processes Simultaneously.

Semaphore acts as a guard | Lock on the Resources.

When ever a process needs to access the resource, it first needs to take permission from the semaphore.

Semaphore give permission to access a resource if resource is free otherwise process has to wait.

The Semaphore is implemented by variables like counter, a waiting list of processes and two methods (eg. functions) Signal & wait. with integer values.



The Semaphore is accessed by only two indivisible operations known as 'wait' and 'signal' operations which is denoted by P and V.

Example:-

→ Process performs 'wait' operation when tries to enter 'critical Section'

→ Semaphore allow process to enter Critical section is not being used by any process other wise deny it

→ The count of the Semaphore is decremented if a process access critical section

→ Initially the count of Semaphore is 1. and if it is accessed then count decremented and become zero.(0).

→ When a process exits the CS, it peforms the Signal operation which is an exit criterion.

In this way, the solution to CS using semaphore Satisfied the designed protocols.

The Semaphore whose value either 0 or 1 is known as Binary Semaphore. This Concept is basically used in mutual-exclusion.

```
do {
wait( Semaphore)
{
    Critical
     Section
}
Signal ( Semaphore )
{ ....
  ....
}
```

When there is more processes^{3} which want to access more available resources say 3 memories unit then the Semaphore is taken to guard all the three memory locations with Value 3. It means that 3 processes at the Same time Can access the Semaphore. After giving the access to 3 third process the value of count become 0.

. This type of Semaphore that takes a value greater than one is known as 'Counting Semaphore'.