# Assignment 6

→ **Aim** :

To implement Dining Philosopher's problem using 'C' in Linux.

→ **Objective** :

- Implement the deadlock-free solution to Dining Philosophers problem to illustrate the problem of deadlock/starvation that can occur when many synchronised threads are competing for limited resources.

→ **Theory** :

A] What is deadlock?

- A set of process is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Because all the processes are waiting, none of them will ever cause any of the events that could wake up any of other members of the set, and all the processes ~~can~~ continue to wait forever.

## Conditions for Deadlock :

A set of processes is deadlocked when :

1. Mutual exclusion Condition : Each resource is either currently assigned to exactly one process or is available.
2. Hold & wait condition : Process currently holding resources granted earlier can request new resources.
3. No preemption required : Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
4. Circular Wait Condition : There must be circular chain of 2 or more ~~resources~~ processes, each of which is waiting for a resource held by the next member of the chain.

---

→ Dining Philosopher's Problem :

There are 5 philosophers present who spend their life in eating & thinking. They share a common ~~circl~~ circular table surrounded by 5 chairs.

At the centre there is a bowl of food & across each philosopher a pair of chopsticks. When a philosopher is thinking, he does not interact.

Whenever a philosopher gets hungry, he tries to pick up the 2 chopsticks closest to him. He may pick up one chopstick at a time. He cannot pick the chopstick that is already in the hand of neighbor. ~~[illegible]~~ He can eat only when he has 2 chopsticks (one in each hand) without releasing the chopstick and starts thinking again.
Problem occurs when all try to keep the chopstick at the same time.

---

→ Semaphore Chopstick

Think : After eating
Eat : Hungry

```
Do {
   wait (chopstick[i]);
   wait (chopstick[(i+1)%5]);
   ...
   ...
   eat
   signal (chopstick[i]);
   signal (chopstick[(i+1)%5]);
   ...
   ...
   think;
 } ....
 while (1);
```

→ **Conclusion :**

**Topic Covered :**

1] Dining Philosopher's Problem
2] Deadlock
3] Mutual Exclusion.

---