Starvation : A thread may never get a chance to run if there is always a higher priority thread running or a same-priority thread that never yields. This situation is known as Starvation

Example On Starvation is Dining Philosophers problem Or Any Multitasking system with any shared data between threads and with poor switch or scheduling algorithm

Starvation can be solved by assign sleep() or yield() to threads with same priority to give chance to other threads to run.

In our Problem Dining Philosophers problem we can solve starvation by Declaring Meal Variable, initialize it with 2 and check if one philosopher is eating and Meal is = 0 it interrupt the philosopher and exit it , also there is a loop to switch the forks with neighbors.

Example of deadlock

```
Semaphore chopstick[5]={1};
Int i;
Void philosopher (i)
{
while (true) { wait(chopstick[i]); // left
chopstick wait(chopstick[(i+1)%5 ]); // right
chopstick
// eat signal(chopstick[i]); // left chopstick
signal(chopstick[(i+1)%5 ]); // right chopstick
// think
}
```

In the above structure, first wait operation is performed on chopstick[i] and chopstick[(i+1) % 5]. This means that the philosopher i has picked up the chopsticks on his sides. Then the eating function is performed.

After that, signal operation is performed on chopstick[i] and chopstick[(i+1) % 5]. This means that the philosopher i has eaten and put down the chopsticks on his sides. Then the philosopher goes back to thinking.

The above solution makes sure that no two neighboring philosophers can eat at the same time. But this solution can lead to a deadlock. This may happen if all the philosophers pick their left chopstick simultaneously. Then none of them can eat and **deadlock** occurs.

Example of deadlock Solution

```
Semaphore chopstick[5]={1};
Int i;
Void philosopher (i)
{
while (true) { if (chopstick[i] &&
chopstick[(i+1)%5]) { wait(chopstick[i]); //
left chopstick wait(chopstick[(i+1)%5 ]); //
right chopstick
// eat signal(chopstick[i]); // left chopstick
signal(chopstick[(i+1)%5 ]); // right chopstick
// think
}
}
```

[if (chopstick[i] && chopstick[(i+1)%5])**]** is the **solution for deadlock** that a philosopher should only be allowed to pick their chopstick if both are available at the same time.

Pseudocode:

```
void philosopher(int i)
{
  While(true)
  {
    take_fork(i); //pick the left fork\     take_fork((i+1)%N); //pick
the right fork     eat();
    put_fork(i);  //put the left fork     put_fork((i+1)%N);   //put the
right fork

  }
}
```

Real world scenario :

There is one room available in the hotel and there are five people and 5 cards.

When a person wants to enter the room in order to sleep, he must have two cards, and one person must enter at time.