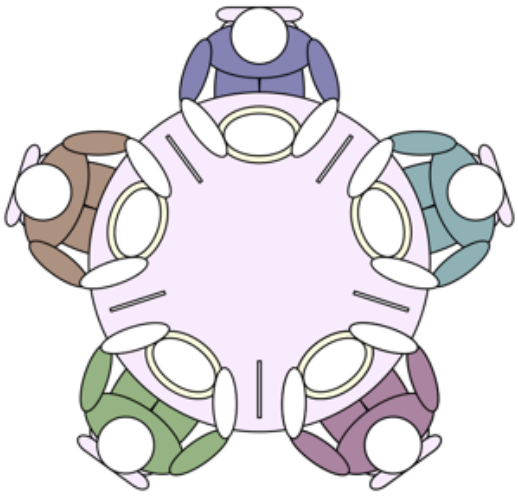
Assignment 6: **Implement the deadlock-free solution to Dining Philosophers problem to illustrate the problem of deadlock and/or starvation that can occur when many synchronized threads are competing for limited resources.**

Name: **Chaitanya Suresh Uge**
Roll Number: **333064**
GR Number: **21910718**
Division: **C** **Batch:** **C3**

Theory:

The Dining Philosopher Problem –

The Dining Philosopher Problem states that K philosophers are seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.



- There are three states of the philosopher: THINKING, HUNGRY, and EATING.
- Here there are two semaphores: Mutex and a semaphore array for the philosophers.
- Mutex is used such that no two philosophers may access the pickup or put down at the same time.
- The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.

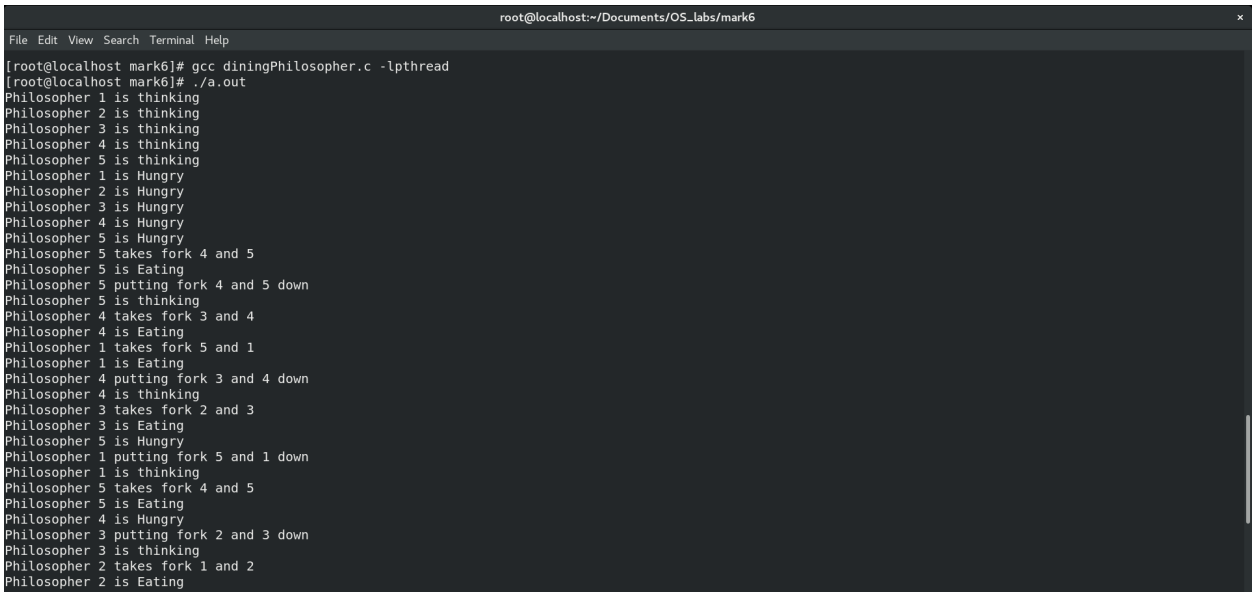
Source Code:

```
1   #include <pthread.h>
2   #include <semaphore.h>
3   #include <stdio.h>
4   #include <unistd.h>
5
6   #define N 5
7   #define THINKING 2
8   #define HUNGRY 1
9   #define EATING 0
10  #define LEFT (phnum + 4) % N
11  #define RIGHT (phnum + 1) % N
12
13  void * philospher(void *num);
14  void take_fork(int);
15  void put_fork(int);
16  void test(int);
17
18  int state[N];
19  int phil[N] = { 0, 1, 2, 3, 4 };
20
21  sem_t mutex;
22  sem_t S[N];
23
24  void test(int phnum)
25  {
26     if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
27     {
28        state[phnum] = EATING;
29        sleep(2);
30        printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
31        printf("Philosopher %d is Eating\n", phnum + 1);
32        sem_post(&S[phnum]);
```

```
33      }
34  }
35
36  // take up chopsticks
37  void take_fork(int phnum)
38  {
39      sem_wait(&mutex);
40
41      state[phnum] = HUNGRY;
42      printf("Philosopher %d is Hungry\n", phnum + 1);
43      // eat if neighbours are not eating
44      test(phnum);
45      sem_post(&mutex);
46      // if unable to eat wait to be signalled
47      sem_wait(&S[phnum]);
48      sleep(1);
49  }
50
51  // put down chopsticks
52  void put_fork(int phnum)
53  {
54      sem_wait(&mutex);
55
56      state[phnum] = THINKING;
57      printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);
58      printf("Philosopher %d is thinking\n", phnum + 1);
59      test(LEFT);
60      test(RIGHT);
61      sem_post(&mutex);
62  }
63
64  void* philosopher(void* num)
65  {
66      while (1)
67      {
68          int* i = num;
69          sleep(1);
70          take_fork(*i);
71          sleep(0);
72          put_fork(*i);
73      }
74  }
75
76  int main()
77  {
78      int i;
79      pthread_t thread_id[N];
80
81      sem_init(&mutex, 0, 1);
82      for (i = 0; i < N; i++)
83          sem_init(&S[i], 0, 0);
84      for (i = 0; i < N; i++)
```

```
85      {
86          pthread_create(&thread_id[i], NULL, philosopher, &phil[i]);
87          printf("Philosopher %d is thinking\n", i + 1);
88      }
89
90      for (i = 0; i < N; i++)
91      {
92          pthread_join(thread_id[i], NULL);
93      }
94
95          return 0;
96  }
```

OUTPUT:



```
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
```