

## PRACTICAL 5

**Aim:** Considered there are N philosophers 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. Write a program to solve the problem using process synchronization technique.

```
GNU nano 8.7.1 philosopher.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define N 5
#define EAT_COUNT 3 // Number of times each philosopher eats

pthread_mutex_t chopstick[N];

void* philosopher(void* num)
{
    int id = *(int*)num;
    int left = id;
    int right = (id + 1) % N;

    // Deadlock prevention: pick lower-numbered chopstick first
    int first = left < right ? left : right;
    int second = left < right ? right : left;

    for (int i = 0; i < EAT_COUNT; i++)
    {
        printf("Philosopher %d is THINKING\n", id);
        sleep(1);

        printf("Philosopher %d is HUNGRY\n", id);

        pthread_mutex_lock(&chopstick[first]);
        pthread_mutex_lock(&chopstick[second]);

        printf("Philosopher %d is EATING (%d)\n", id, i + 1);
        sleep(2);

        pthread_mutex_unlock(&chopstick[second]);
        pthread_mutex_unlock(&chopstick[first]);

        printf("Philosopher %d Finished EATING (%d)\n\n", id, i + 1);
    }

    return NULL;
}

int main()
{
    pthread_t tid[N];
    int phil_id[N];

```

```
GNU nano 8.7.1 philosopher.c
    int first = left < right ? left : right;
    int second = left < right ? right : left;

    for (int i = 0; i < EAT_COUNT; i++)
    {
        printf("Philosopher %d is THINKING\n", id);
        sleep(1);

        printf("Philosopher %d is HUNGRY\n", id);

        pthread_mutex_lock(&chopstick[first]);
        pthread_mutex_lock(&chopstick[second]);

        printf("Philosopher %d is EATING (%d)\n", id, i + 1);
        sleep(2);

        pthread_mutex_unlock(&chopstick[second]);
        pthread_mutex_unlock(&chopstick[first]);

        printf("Philosopher %d finished EATING (%d)\n\n", id, i + 1);
    }

    return NULL;
}

int main()
{
    pthread_t tid[N];
    int phil_id[N];

    for (int i = 0; i < N; i++)
        pthread_mutex_init(&chopstick[i], NULL);

    for (int i = 0; i < N; i++)
    {
        phil_id[i] = i;
        pthread_create(&tid[i], NULL, philosopher, &phil_id[i]);
    }

    for (int i = 0; i < N; i++)
        pthread_join(tid[i], NULL);

    printf("All philosophers have finished eating.\n");
    return 0;
}
```

## Output:

```
ASUS@LAPTOP-821G7DSB MINGW64 ~  
$ nano philosopher.c  
  
ASUS@LAPTOP-821G7DSB MINGW64 ~  
$ gcc philosopher.c -o philosopher  
  
ASUS@LAPTOP-821G7DSB MINGW64 ~  
$ ./philosopher  
Philosopher 3 is THINKING  
Philosopher 0 is THINKING  
Philosopher 2 is THINKING  
Philosopher 1 is THINKING  
Philosopher 4 is THINKING  
Philosopher 0 is HUNGRY  
Philosopher 0 is EATING (1)  
Philosopher 3 is HUNGRY  
Philosopher 3 is EATING (1)  
Philosopher 2 is HUNGRY  
Philosopher 4 is HUNGRY  
Philosopher 1 is HUNGRY  
Philosopher 0 finished EATING (1)  
  
Philosopher 0 is THINKING  
Philosopher 3 finished EATING (1)  
  
Philosopher 3 is THINKING  
Philosopher 4 is EATING (1)  
Philosopher 2 is EATING (1)  
Philosopher 0 is HUNGRY  
Philosopher 3 is HUNGRY  
Philosopher 4 finished EATING (1)  
  
Philosopher 4 is THINKING  
Philosopher 3 is EATING (2)  
Philosopher 1 is EATING (1)  
Philosopher 2 finished EATING (1)  
  
Philosopher 2 is THINKING  
Philosopher 4 is HUNGRY  
Philosopher 2 is HUNGRY  
Philosopher 3 finished EATING (2)  
  
Philosopher 1 finished EATING (1)  
  
Philosopher 1 is THINKING  
Philosopher 0 is EATING (2)  
Philosopher 3 is THINKING  
Philosopher 2 is EATING (2)  
Philosopher 1 is HUNGRY  
Philosopher 3 is HUNGRY
```

```
M ~
Philosopher 2 finished EATING (1)

Philosopher 2 is THINKING
Philosopher 4 is HUNGRY
Philosopher 2 is HUNGRY
Philosopher 3 finished EATING (2)

Philosopher 1 finished EATING (1)

Philosopher 1 is THINKING
Philosopher 0 is EATING (2)
Philosopher 3 is THINKING
Philosopher 2 is EATING (2)
Philosopher 1 is HUNGRY
Philosopher 3 is HUNGRY
Philosopher 3 is EATING (3)
Philosopher 1 is EATING (2)
Philosopher 0 finished EATING (2)

Philosopher 0 is THINKING
Philosopher 2 finished EATING (2)

Philosopher 2 is THINKING
Philosopher 2 is HUNGRY
Philosopher 0 is HUNGRY
Philosopher 3 finished EATING (3)

Philosopher 4 is EATING (2)
Philosopher 2 is EATING (3)
Philosopher 1 finished EATING (2)

Philosopher 1 is THINKING
Philosopher 1 is HUNGRY
Philosopher 4 finished EATING (2)

Philosopher 4 is THINKING
Philosopher 1 is EATING (3)
Philosopher 2 finished EATING (3)

Philosopher 4 is HUNGRY
Philosopher 1 finished EATING (3)

Philosopher 0 is EATING (3)
Philosopher 0 finished EATING (3)

Philosopher 4 is EATING (3)
Philosopher 4 finished EATING (3)

All philosophers have finished eating.

ASUS@LAPTOP-82JG7DSB MINGW64 ~
$ nano philosopher.c

ASUS@LAPTOP-82JG7DSB MINGW64 ~
$
```

