

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 anyone of its adjacent followers but not both. Write a program to solve the problem using process synchronization technique.

```

// Philosopher.h
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

#define N 5
#define EAT_COUNT 5 // number of times each philosopher eats

pthread_mutex_t chopsticks[N];

void* philosopher(void* num)
{
    int id = *((int*)num);
    int left = id;
    int right = (id + 2) % 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(2);

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

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

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

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

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

    return NULL;
}

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

    for (int i = 0; i < N; i++)
    {
        pthread_mutex_t* tchopsticks[i] = NULL;
    }

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

    return 0;
}

```

```
gcc main.c -o phiTester.exe
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
  
int id = 0;  
int left = 0;  
int right = (id + 1) % N;  
  
// deadlock prevention: pick lower-numbered chapterick first  
let flock = {left < right ? left : right};  
let second = left > right ? right : left;  
  
for (int i = 0; i < EAT_COUNT; i++)  
{  
    printf("philosopher %d is THINKING\n", id);  
    sleep(2);  
  
    pthread_mutex_lock(&chapterick[flock]);  
    pthread_mutex_lock(&chapterick[second]);  
  
    printf("philosopher %d is EATING (%d)\n", id, i + 1);  
    sleep(2);  
  
    pthread_mutex_unlock(&chapterick[second]);  
    pthread_mutex_unlock(&chapterick[flock]);  
  
    printf("philosopher %d Finished EATING (%d)\n", id, i + 1);  
  
    return NULL;  
}  
  
void main()  
{  
    pthread_t t[N];  
    int phil_id[N];  
  
    for (int i = 0; i < N; i++)  
        pthread_create(&t[i], NULL,  
            pthread_main, &i);  
  
    for (int i = 0; i < N; i++)  
    {  
        phil_id[i] = i;  
        pthread_join(t[i], NULL);  
    }  
  
    for (int i = 0; i < N; i++)  
        pthread_join(t[i], NULL);  
  
    printf("all philosophers have finished eating.\n");  
    return 0;  
}
```

The screenshot shows the execution of the program. The output indicates that all philosophers have successfully completed their eating process without encountering a deadlock.

OUTPUT:

```
corinna@kali:~/Desktop$ gcc -o philosopher.c
corinna@kali:~/Desktop$ gcc philosopher.c -o philosopher
corinna@kali:~/Desktop$ ./philosopher
Philosopher 0 is THINKING
Philosopher 1 is THINKING
Philosopher 2 is THINKING
Philosopher 3 is THINKING
Philosopher 4 is THINKING
Philosopher 4 is HUNGRY
Philosopher 4 is EATING (1)
Philosopher 2 is HUNGRY
Philosopher 3 is HUNGRY
Philosopher 1 is HUNGRY
Philosopher 0 is HUNGRY
Philosopher 2 is EATING (1)
Philosopher 4 finished EATING (1)
Philosopher 2 finished EATING (1)
Philosopher 3 is EATING (1)
Philosopher 1 is EATING (1)
Philosopher 4 is THINKING
Philosopher 2 is THINKING
Philosopher 4 is HUNGRY
Philosopher 2 is HUNGRY
Philosopher 0 is EATING (1)
Philosopher 1 finished EATING (1)
Philosopher 1 is THINKING
Philosopher 2 is EATING (2)
Philosopher 3 finished EATING (1)
Philosopher 3 is THINKING
Philosopher 2 is HUNGRY
Philosopher 1 is HUNGRY
Philosopher 4 is EATING (2)
Philosopher 2 finished EATING (2)
Philosopher 1 is EATING (2)
Philosopher 0 finished EATING (1)
Philosopher 2 is THINKING
Philosopher 0 is THINKING
Philosopher 0 is HUNGRY
Philosopher 2 is HUNGRY
Philosopher 1 finished EATING (2)
Philosopher 1 is THINKING
Philosopher 3 is EATING (2)
Philosopher 4 finished EATING (2)
Philosopher 4 is THINKING
```

```
Philosopher 3 finished EATING (1)
Philosopher 1 is THINKING
Philosopher 3 is HUNGRY
Philosopher 2 is HUNGRY
Philosopher 4 is EATING (2)
Philosopher 2 finished EATING (2)
Philosopher 1 is EATING (2)
Philosopher 0 finished EATING (1)
Philosopher 2 is THINKING
Philosopher 0 is THINKING
Philosopher 0 is HUNGRY
Philosopher 2 is HUNGRY
Philosopher 1 finished EATING (2)
Philosopher 2 is THINKING
Philosopher 2 is EATING (2)
Philosopher 4 finished EATING (2)
Philosopher 4 is THINKING
Philosopher 0 is EATING (2)
Philosopher 3 is HUNGRY
Philosopher 4 is HUNGRY
Philosopher 1 finished EATING (2)
Philosopher 2 is EATING (2)
Philosopher 3 is THINKING
Philosopher 0 is HUNGRY
Philosopher 2 is HUNGRY
Philosopher 1 finished EATING (3)
Philosopher 2 is EATING (3)
Philosopher 1 is EATING (3)
Philosopher 4 finished EATING (3)
Philosopher 3 finished EATING (3)
Philosopher 1 finished EATING (3)
Philosopher 0 is EATING (3)
Philosopher 0 finished EATING (3)
All philosophers have finished eating.
corinna@kali:~/Desktop$ gcc -o philosopher.c
corinna@kali:~/Desktop$ gcc philosopher.c -o philosopher
corinna@kali:~/Desktop$ ./philosopher
```