Name	Manish Shashikant Jadhav
UID no.	2023301005

Experiment 7

AIM :

C program implementing the solution to the Dining Philosopher Problem. The Dining Philosopher Problem states that there are five philosophers which do two thinks: think and eat. They share a table having a chair for each one of them. In the center of the table there is a bowl of rice and the table is laid with 5 single chopsticks .

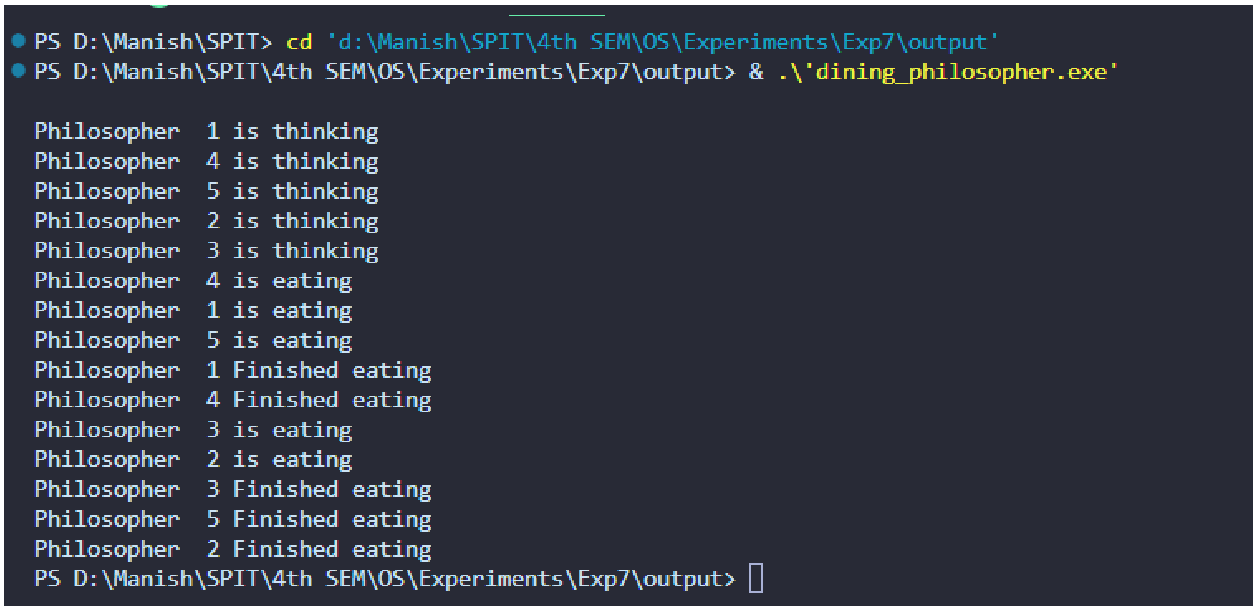
When a philosopher thinks, he does not interact with others. When he gets hungry, he tries to pick up the two chopsticks that are near to him. For example, philosopher 1 will try to pick chopsticks 1 and 2. But the philosopher can pickup only one chopstick at a time. He can not take a chopstick that is already in the hands of his neighbor. The philosopher stars to eat when he has both his chopsticks in his hand. After eating the philosopher puts down both the chopsticks and starts to think again.

Discussion & Output:

Program:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#define num_philopsophers 5
#define num_chopsticks 5

void dine(int n);
pthread_t philosopher[num_philopsophers];
pthread_mutex_t chopstick[num_chopsticks];

int main()
{
    // Define msg and status_message
    int status_message;
    void *msg;

    // Initialise the semaphore array
    for (int i = 1; i <= num_chopsticks; i++)
    {
        status_message = pthread_mutex_init(&chopstick[i], NULL);

        // Checking if the mutex was initialised successfully
        if (status_message == -1)
        {
            printf("\n Mutex initialization failed");
            exit(1);
        }
    }

    // Run the philosopher Threads using *dine() function
    for (int i = 1; i <= num_philopsophers; i++)
    {
        status_message = pthread_create(&philosopher[i], NULL,
(void *)dine, (int *)i);
        if (status_message != 0)
        {
            printf("\n Thread creation error \n");
            exit(1);
        }
    }
    // Wait for all philosophers threads to complete executing
    // (finish dining) before closing the program
    for (int i = 1; i <= num_philopsophers; i++)
    {
        status_message = pthread_join(philosopher[i], &msg);
        if (status_message != 0)
        {
            printf("\n Thread join failed \n");
            exit(1);
        }
    }

    // Destroy the chopstick Mutex array
    for (int i = 1; i <= num_chopsticks; i++)
    {
        status_message = pthread_mutex_destroy(&chopstick[i]);
        if (status_message != 0)
        {
            printf("\n Mutex Destroyed \n");
            exit(1);
        }
    }
    return 0;
}

// dine method
void dine(int n)
{
    printf("\nPhilosopher % d is thinking ", n);

    // picking up the left chopstick (wait)
    pthread_mutex_lock(&chopstick[n]);

    // picking up the right chopstick (wait)
    pthread_mutex_lock(&chopstick[(n + 1) % num_chopsticks]);

    // both chopstick picked now starts eating

    printf("\nPhilosopher % d is eating ", n);
    sleep(3);

    // places the left chopstick down (signal)
    pthread_mutex_unlock(&chopstick[n]);

    // places the  the right chopstick down (signal)
    pthread_mutex_unlock(&chopstick[(n + 1) % num_chopsticks]);

    //  eating finishes
    printf("\nPhilosopher % d Finished eating ", n);
}
```

Output:



```
● PS D:\Manish\SPIT> cd 'd:\Manish\SPIT\4th SEM\OS\Experiments\Exp7\output'
● PS D:\Manish\SPIT\4th SEM\OS\Experiments\Exp7\output> & .\'dining_philosopher.exe'

  Philosopher  1 is thinking
  Philosopher  4 is thinking
  Philosopher  5 is thinking
  Philosopher  2 is thinking
  Philosopher  3 is thinking
  Philosopher  4 is eating
  Philosopher  1 is eating
  Philosopher  5 is eating
  Philosopher  1 Finished eating
  Philosopher  4 Finished eating
  Philosopher  3 is eating
  Philosopher  2 is eating
  Philosopher  3 Finished eating
  Philosopher  5 Finished eating
  Philosopher  2 Finished eating
  PS D:\Manish\SPIT\4th SEM\OS\Experiments\Exp7\output> 
```

CONCLUSION:	Hence, by completing this experiment I came to know about implementing the solution to the Dining Philosopher Problem.