**29) Write a C program to simulate the solution of Classical Process Synchronization Problem.**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_PHILOSOPHERS 5

// Semaphores

sem\_t chopsticks[NUM\_PHILOSOPHERS];

void \*philosopher(void \*num) {

int id = \*(int \*)num;

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

sleep(1);

// Pick up chopsticks

sem\_wait(&chopsticks[id]);

sem\_wait(&chopsticks[(id + 1) % NUM\_PHILOSOPHERS]);

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

sleep(1);

// Put down chopsticks

sem\_post(&chopsticks[id]);

sem\_post(&chopsticks[(id + 1) % NUM\_PHILOSOPHERS]);

printf("Philosopher %d is done eating and thinking again.\n", id);

return NULL;

}

int main() {

pthread\_t threads[NUM\_PHILOSOPHERS];

int ids[NUM\_PHILOSOPHERS];

// Initialize semaphores

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

sem\_init(&chopsticks[i], 0, 1);

}

// Create philosopher threads

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

ids[i] = i;

pthread\_create(&threads[i], NULL, philosopher, &ids[i]);

}

// Wait for all threads to complete

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

pthread\_join(threads[i], NULL);

}

// Destroy semaphores

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

sem\_destroy(&chopsticks[i]);

}

return 0;

}

Output:

