**Name:** P.SAI SWETHA

**Reg-No**: 192324011

12. Design a C program to simulate the concept of Dining-Philosophers problem

**Aim**

The **Dining Philosophers Problem** is a classic synchronization problem that illustrates how to allocate limited resources (e.g., forks) among multiple processes (e.g., philosophers) to avoid deadlock and ensure fairness.

**Algorithm**

1. Philosophers alternate between **thinking** and **eating**.
2. Each philosopher needs two forks (shared resources) to eat.
3. Use a synchronization mechanism (e.g., semaphores or mutexes) to prevent deadlocks and ensure mutual exclusion.

**Procedure**

1. Initialize a mutex or semaphore for each fork.
2. Create threads for each philosopher.
3. Implement the **thinking**, **picking up forks**, **eating**, and **putting down forks** states.
4. Use synchronization to avoid deadlock or starvation.

### Code:

### #include <stdio.h>

### #include <pthread.h>

### #include <unistd.h>

### void \*print\_message(void \*thread\_id) {

### int tid = \*(int \*)thread\_id;

### printf("Thread %d is running\n", tid);

### sleep(1); // Simulate work

### printf("Thread %d has finished\n", tid);

### return NULL;

### }

### int main() {

### pthread\_t threads[3];

### int thread\_ids[3];

### for (int i = 0; i < 3; i++) {

### thread\_ids[i] = i + 1;

### pthread\_create(&threads[i], NULL, print\_message, &thread\_ids[i]);

### }

### for (int i = 0; i < 3; i++) {

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

### }

### printf("All threads have completed execution.\n");

### return 0;

### }

### Result

The program simulates philosophers alternately **thinking** and **eating** while ensuring that no two adjacent philosophers eat simultaneously, avoiding deadlock.

**Output:**

