**Name:** P.SAI SWETHA

**Reg-No**: 192324011

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

**Aim:**

To simulate the solution to the Classical Process Synchronization Problem (e.g., Producer-Consumer or Dining Philosophers) using C programming and demonstrate the correct functioning of process synchronization.

**Algorithm (Dining Philosophers Example):**

1. Initialize the state of philosophers as "thinking."
2. Use semaphores to control access to shared resources (chopsticks).
3. Define pickup() and putdown() functions to manage chopsticks.
4. A philosopher alternates between thinking and eating.
5. Ensure no deadlock or starvation occurs using a synchronization mechanism.

**Procedure:**

1. Create threads to represent philosophers.
2. Use semaphores for chopstick access.
3. Implement synchronization logic to prevent deadlock (e.g., wait-and-signal operations).
4. Run the program and observe how philosophers alternate between thinking and eating.

**Code:**

### #include <pthread.h>

### #include <semaphore.h>

### #include <stdio.h>

### #include <unistd.h>

### #define N 5

### sem\_t chopstick[N];

### pthread\_t philosopher[N];

### void\* dine(void\* arg) {

### int id = \*(int\*)arg;

### while (1) {

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

### sleep(1);

### sem\_wait(&chopstick[id]);

### sem\_wait(&chopstick[(id + 1) % N]);

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

### sleep(1);

### sem\_post(&chopstick[id]);

### sem\_post(&chopstick[(id + 1) % N]);

### printf("Philosopher %d finished eating and starts thinking.\n", id);

### }

### }

### int main() {

### int id[N];

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

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

### id[i] = i;

### }

### for (int i = 0; i < N; i++)

### pthread\_create(&philosopher[i], NULL, dine, &id[i]);

### for (int i = 0; i < N; i++)

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

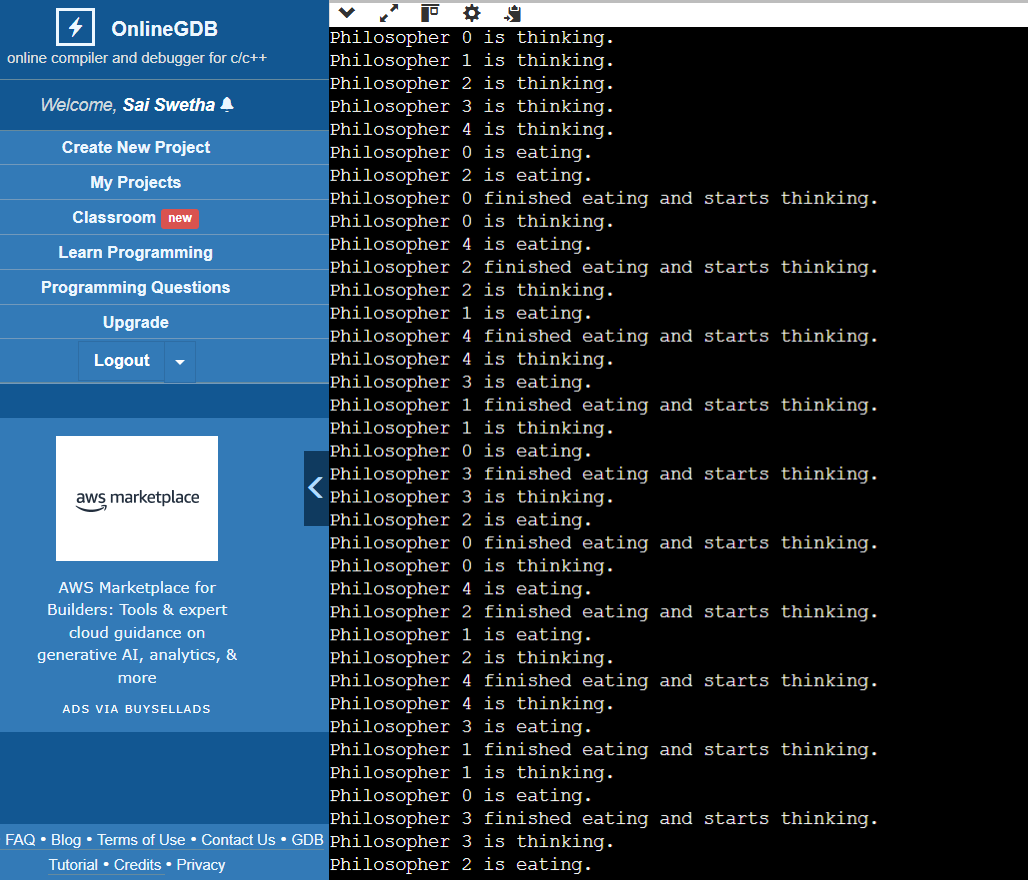
### return 0;

### }

### Result:

The output of the program demonstrates that each philosopher alternates between thinking and eating, ensuring proper synchronization without deadlock or starvation.

**Output:**

****