#### Dining Philosophers Problem

# Aim:

To implement a program for solving Dining Philosophers problem using Semaphores.

# Description:

The Dining Philosophers problem is a classic synchronization problem in computer science and concurrency theory. It illustrates the challenges of resource allocation and deadlock prevention in a concurrent system.

The philos() function represents the behavior of each philosopher. When a philosopher is thinking, they need to acquire both the left and right chopsticks before they can eat. This is done by waiting on the semaphores representing the chopsticks (sem\_wait).

After acquiring both chopsticks, the philosopher can eat for a period of time (in this case, simulated by sleep(2)).

After finishing eating, the philosopher releases both chopsticks (sem\_post), allowing other philosophers to use them.

Finally, the philosopher goes back to thinking.

In the main function, the code initializes the semaphores for the chopsticks, setting their initial value to 1 (available).

Then, it creates 5 threads (one for each philosopher) using pthread\_create, passing the philosopher number (arg[i]) to the philos() function.

Finally, it waits for all philosopher threads to finish using pthread\_join.

This ensures that no two neighboring philosophers can eat simultaneously, thus preventing deadlock

**Algorithm:**

> start

> Define the number of philosophers (N = 5).

> Initialize an array of semaphores sticks with size N.

> Define a function philos that takes an integer n as a parameter:

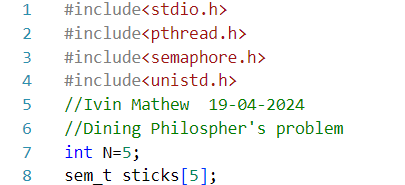
* Convert n to an integer i.
* Wait (lock) on the left stick semaphore (sticks[i]).
* Wait (lock) on the right stick semaphore (sticks[(i+1)%N]).
* Print that philosopher i has taken the left stick.
* Print that philosopher i has taken the right stick.
* Print that philosopher i is eating.
* Sleep for 2 seconds to simulate eating.
* Print that philosopher i has finished eating.
* Post (unlock) the left stick semaphore.
* Post (unlock) the right stick semaphore.
* Print that philosopher i is thinking.

> In the main function:

* Initialize an array arg with size N.
* Initialize an array of threads t with size N.
* Initialize each semaphore in sticks to 1.
* For each philosopher i from 0 to N-1:
  + Set arg[i] to i.
  + Create a new thread t[i] and assign it to execute the philos function with argument arg[i].
* For each thread i from 0 to N-1:
  + Join thread t[i] to ensure the main program waits for the thread to finish.

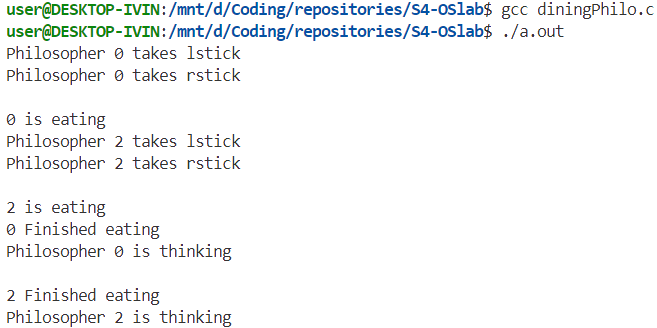
> end

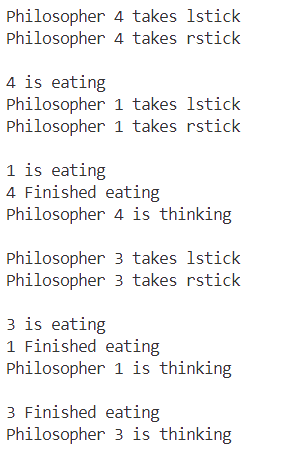
# Code



# 

# Sample Output





# Result

The program has been executed and output has been verified