DINING PHILOSOPHERS PROBLEM

BY :

SAI ROHAN PAWAR (21114088) (2Y)

LAKSHIT SHARMA (20115058) (3Y)

PROBLEM STATEMENT

- Dining Philosophers problem is a classic synchronization problem. The problem involves a group of philosophers sitting around a circular table with a fork on either side of each philosopher. The philosophers alternate between thinking and eating, but they need two forks to eat. The challenge is to develop a solution to ensure that each philosopher can eat without getting into a deadlock.
- We need to ensure that no adjacent philosophers should take the fork at same time to prevent deadlock.

Semaphore function

```
#include <iostream>
#include <stdlib.h>
#include <pthread.h>
using namespace std;
#define n 10
//Here Each of the fork is taken as a binary semaphore
int fork[n];
//here 1 indicates that fork is available, 0 indicates that fork is taken;
int phil[n] ;

void sleep(double d){
  while(d--);
}
void wait(int* fork1, int* fork2){

 //Wait till both the forks are not taken
  while(*fork1 == 0 || *fork2 == 0);

  *fork1 -= 1;
  *fork2 -= 1;
}

void signal(int* fork1, int* fork2){
  *fork1 += 1;
  *fork2 += 1;
}
```

Philosopher function

```
void* philospher(void* num)
{
  int *ID = (int*)num;
  int i = *ID;
  //here i is Philosopher ID
    while (1) {
        cout<<"Philosopher "<<i<<" is Hungry"<<endl;
        wait(fork+i, fork+((i-1)%n));
        cout<<"Philosopher "<<i<<" is Eating"<<endl;
        //its the critical section
        sleep(200000000);

        signal(fork + i,fork+((i-1)%n));
        cout<<"Philosopher "<<i<<" has finished Eating"<<endl;
        cout<<"Forks "<<i<<" & "<<(i-1)%n<<" are free"<<endl;
        //eating is done, wait for random time before hungry again;
        cout<<"Philosopher "<<i<<" is Thinking"<<endl;
        double waiter = rand() % 200000000 + 200000000;
        sleep(waiter);

    }
}

```

Main function

```
50. int main() {
51.     pthread_t thread_id[n];
52.      for (int i = 0; i < n; i++){
53.         // create the philosopher processes
54.         fork[i]=1;
55.         phil[i]=i;
56.         pthread_create(&thread_id[i], NULL, philospher, &phil[i]);
57.      }
58.         // starting theprocesses
59.         for (int i = 0; i < n; i++)
60.         pthread_join(thread_id[i], NULL);
61. }
```

OUTPUT

stdout

```
Philosopher 6 is Hungry
Philosopher 6 is Eating
Philosopher 7 is Hungry
Philosopher 8 is Hungry
Philosopher 8 is Eating
Philosopher 9 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 is Hungry
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 0 is Hungry
Philosopher 8 has finished Eating
Forks 8 & 7 are free
Philosopher 8 is Thinking
Philosopher 4 has finished Eating
Forks 4 & 3 are free
Philosopher 4 is Thinking
Philosopher 6 has finished Eating
Forks 6 & 5 are free
Philosopher 6 is Thinking
Philosopher 2 has finished Eating
Forks 2 & 1 are free
Philosopher 2 is Thinking
Philosopher 8 is Hungry
Philosopher 8 is Eating
Philosopher 4 is Hungry
Philosopher 4 is Eating
Philosopher 6 is Hungry
Philosopher 6 is Eating
```