Task 7. [CO3]: Process Synchronization needs to be implemented to prevent data inconsistency among processes, process deadlocks.

Aim

To write a c program for process synchronization to implemented to prevent data inconsistency among processes, process deadlocks

program

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[N];
void test(int phnum)
{
        if (state[phnum] == HUNGRY
                && state[LEFT] != EATING
                && state[RIGHT] != EATING) {
                // state that eating
                state[phnum] = EATING;
                sleep(2);
                printf("Philosopher %d takes fork %d and %d\n",
                                        phnum + 1, LEFT + 1, phnum + 1);
                printf("Philosopher %d is Eating\n", phnum + 1);
                // sem_post(&S[phnum]) has no effect
                // during takefork
                // used to wake up hungry philosophers
                // during putfork
                sem_post(&S[phnum]);
        }
}
// take up chopsticks
void take_fork(int phnum)
{
        sem_wait(&mutex);

        // state that hungry
        state[phnum] = HUNGRY;
        printf("Philosopher %d is Hungry\n", phnum + 1);
        // eat if neighbours are not eating
        test(phnum);
        sem_post(&mutex);
        // if unable to eat wait to be signalled
```

```c
        sem_wait(&S[phnum]);
        sleep(1);
}
// put down chopsticks
void put_fork(int phnum)
{
        sem_wait(&mutex);
        // state that thinking
        state[phnum] = THINKING;
        printf("Philosopher %d putting fork %d and %d down\n",
                phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is thinking\n", phnum + 1);
        test(LEFT);
        test(RIGHT);
        sem_post(&mutex);
}
void* philospher(void* num)
{
        while (1) {
                int* i = num;
                sleep(1);
                take_fork(*i);
                sleep(0);
                put_fork(*i);
        }
}
int main()
{
        int i;
        pthread_t thread_id[N];
        // initialize the semaphores
        sem_init(&mutex, 0, 1);
        for (i = 0; i < N; i++)
                sem_init(&S[i], 0, 0);
        for (i = 0; i < N; i++) {
                // create philosopher processes
                pthread_create(&thread_id[i], NULL,
                                        philospher, &phil[i]);
                printf("Philosopher %d is thinking\n", i + 1);
        }
        for (i = 0; i < N; i++)
                pthread_join(thread_id[i], NULL);
}
```