CS4315 Operating Systems
Lab Assignment 4

Dining Philosopher's problem is a famous problem in OS. A deadlock may happen when all philosophers want to start eating at the same time and pick up one chopstick and wait for the other chopstick. We can use semaphores to simulate the availability of chopsticks.

To prevent the deadlock, in class we have discussed three solutions:

1. Allow at most 4 philosophers to be sitting simultaneously at the table.
2. Allow a philosopher to pick up her chopsticks only if both are available
3. Use an asymmetric solution: an odd-numbered philosopher picks up first the left chopstick and then the right chopstick. Even-numbered philosopher picks up first the right chopstick and then the left chopstick.

The following program can lead to a deadlock.

- Run the program and observe the deadlock condition.
- Based on the program, please implement the above three solutions to prevent the deadlock. Each solution should be saved as an individual C program (e.g. solution1.c, solution2.c, and solution3.c). Include necessary comments in your programs.

Submission: solution2.c and solution3.c on Blackboard

```c
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>

#define N 5 //the number of philosophers

sem_t S[N]; //semaphores for chopsticks

void * philospher(void *num);
void take_chopsticks(int);
void put_chopsticks(int);

int phil_num[N]={0,1,2,3,4}; //philosopher ID

int main()
{
    int i;
    pthread_t thread_id[N];


    for(i=0;i<N;i++)
        sem_init(&S[i],0,1);

    for(i=0;i<N;i++)
        pthread_create(&thread_id[i],NULL,philospher,&phil_num[i]);

    for(i=0;i<N;i++)
        pthread_join(thread_id[i],NULL);
}
```

```
void *philospher(void *num)
{
    while(1)
    {
        int *i = num;
        take_chopsticks(*i);
        put_chopsticks(*i);
    }
}


void take_chopsticks(int ph_num)
{

    printf("Philosopher %d is Hungry\n",ph_num);


    sem_wait(&S[ph_num]);  //take the left chopstick
    printf("Philosopher %d takes chopstick %d \n",ph_num, ph_num);

    sleep(1);

    sem_wait (&S[(ph_num+1)%N]);  //take the right chopstick
    printf("Philosopher %d takes chopstick %d \n",ph_num,(ph_num+1)%N);


    printf("Philosopher %d is eating\n",ph_num);
    sleep(1);

 }


void put_chopsticks(int ph_num)
{

   sem_post (&S[ph_num]); //put the left chopstick
   printf("Philosopher %d putting chopstick %d \n",ph_num, ph_num);

   sleep(1);

   sem_post (&S[(ph_num+1)%N]); //put the right chopstick
   printf("Philosopher %d putting chopstick %d \n",ph_num,(ph_num+1)%N);


   printf("Philosopher %d is thinking\n",ph_num);
   sleep(1);

}
```