UIT 1512 – Operating Systems Lab

Process Synchronization problem

Experiment No: 7 **Date: 08.09.2021**

Reg No.: 195002106

Aim:

To Implement the Dining Philosophers Problem of process synchronization.

Problem and Concept description:

Process Synchronization is a process management in OS. In order to perform multitasking, processes can execute concurrently, the process may be interrupted at any time, partially completing execution. There are several classical process-synchronization problems are used to test virtually every new proposed synchronization algorithm such as Bounded-Buffer Problem, Readers-Writers Problem and Dining-Philosophers Problem.

Problem Description:

The dining philosopher's problem states that there are 5 philosophers sharing a circular table and they eat and think alternatively. There is a bowl of rice for each of the philosophers and 5 chopsticks. A philosopher needs both their right and left chopstick to eat. A hungry philosopher may only eat if there are both chopsticks available. Otherwise, a philosopher puts down their chopstick and begin thinking again.

Program:

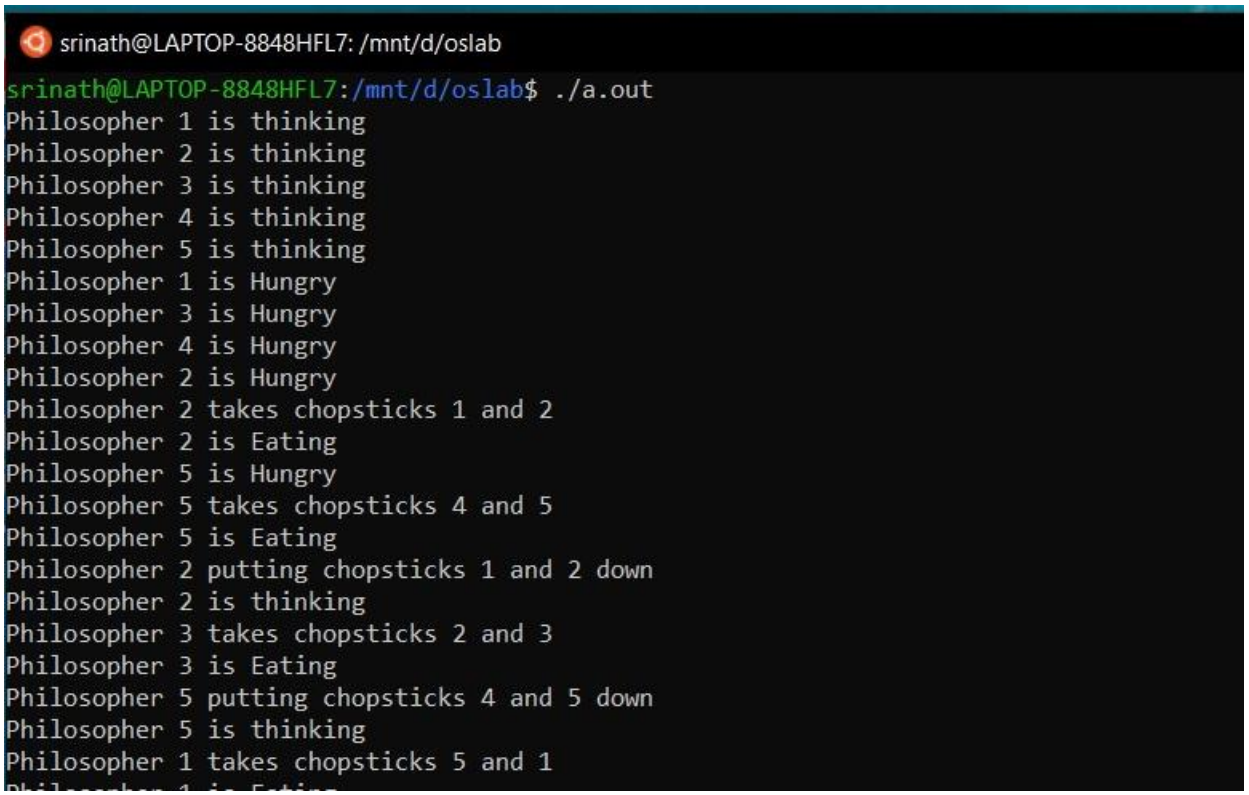
Code:

```
#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 chopsticks %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
        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 chopsticks %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* philosopher(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,
                                        philosopher, &phil[i]);

                printf("Philosopher %d is thinking\n", i + 1);
        }

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

                pthread_join(thread_id[i], NULL);

    return 0;
}
```

Output:



Result:

The implement of the dining philosophers' problem was checked and verified.