Experiment Number: 05

TITLE: Dining Philosophers Problem (Using Semaphore or mutex)

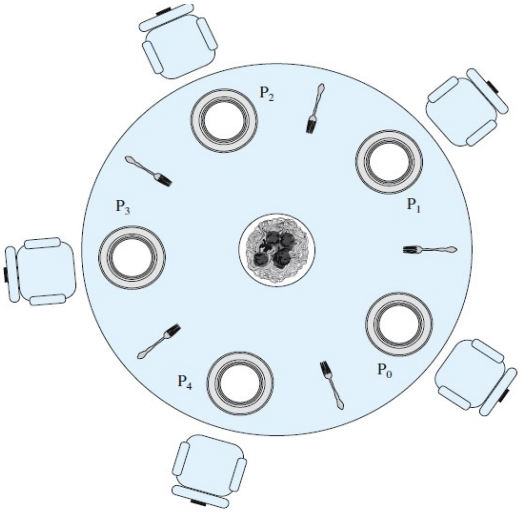
OBJECTIVE:

1. To understand the deadlock and starvation problem using programming.
2. To study dining philosophers problem of operating system.

THEORY:

The Dining Philosophers Problem

The dining philosophers problem was introduced by Dijkstra. Five philosophers live in a house, where a table is laid for them. The life of each philosopher consists principally of thinking and eating, and through years of thought, all of the philosophers had agreed that the only food that contributed to their thinking efforts was spaghetti. Due to a lack of manual skill, each philosopher requires two forks to eat spaghetti.
The eating arrangements are simple: a round table on which is set a large serving bowl of spaghetti, five plates, one for each philosopher, and five forks. A philosopher wishing to eat goes to his or her assigned place at the table and, using the two forks on either side of the plate, takes and eats some spaghetti.
The problem: devise a ritual (algorithm) that will allow the philosophers to eat. The algorithm must satisfy mutual exclusion (no two philosophers can use the same fork at the same time) while avoiding deadlock and starvation. This problem may not seem important or relevant in itself. However, it does illustrate basic problems in deadlock and starvation. Furthermore, attempts to develop solutions reveal many of the difficulties in concurrent programming. In addition, the dining philosophers problem can be seen as representative of problems dealing with the coordination of shared resources, which may occur when an application includes concurrent threads of execution. Accordingly, this problem is a standard test case for evaluating approaches to synchronization.



Dining Arrangement for Philosophers

Solution Using Semaphores

Each philosopher picks up first the fork on the left and then the fork on the right. After the philosopher is finished eating, the two forks are replaced on the table.

```
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
        while (true)
        {
                think();
                wait (fork[i]);
                wait (fork [(i+1) mod 5]);
                eat();
                signal(fork [(i+1) mod 5]);
                signal(fork[i]);
        }
}
void main()
{
        parbegin (philosopher (0), philosopher (1), philosopher (2), philosopher (3),
        philosopher (4));
}
```

ALGORITHM:

PROGRAM (With output and Comments):

CONCLUSION (At least three points):

ASSIGNMENTS:

1. What is the difference between mutex and binary semaphore?
2. How many number of maximum philosophers will this problem have?

REFERENCES:

1. "Beginning Linux Programming" by Neil Mathew and Richard Stones, Wrox Publications.
2. "Operating System Internals and Design Implementation" by William Stallings, Pearson Education.

Tushar B Kute
http://tusharkute.com