**

*National University of Computer & Emerging Sciences*

*(FAST-NU)*

*Operating Systems Project Report*

*[ Project Name:Dining Philosophers Problem]*

*Instructor*

*Miss Sumaiyah.*

*Project Team*

*Muhammad Yasir Javed (16k-3614)*

*Abdul Mannan (16k-3620)*

*Imtiaz Ali (16k-3612)*

*Submission Date*

*03-05-2018*

1. **Project Description**

In computer science, the dining philosophers problem is an example problem often used in concurrent algorithm design to illustrate synchronization issues and techniques for resolving them.

It was originally formulated in 1965 by Edsger Dijkstra as a student exam exercise, presented in terms of computers competing for access to tape drive peripherals. Soon after, Tony Hoare gave the problem its present formulation

1. **Problem Statement**

Five silent philosophers sit at a round table around a bowl of spaghetti. chopsticks are placed between each pair of adjacent philosophers.

Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when they have both left and right chopsticks . Each chopstick can be held by only one philosopher and so a philosopher can use the chopstick only if it is not being used by another philosopher. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others. A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both chopsticks .

Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed.

The problem is how to design a discipline of behaviour (a concurrent algorithm) such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think .

1. **Procedure and Method/implementation (You applied)**

The problem was designed to illustrate the challenges of avoiding [deadlock](https://en.wikipedia.org/wiki/Deadlock), a system state in which no progress is possible. To see that a proper solution to this problem is not obvious, consider a proposal in which each philosopher is instructed to behave as follows:

* think until the left stick is available; when it is, pick it up;
* think until the right stick is available; when it is, pick it up;
* when both sticks are held, eat for a fixed amount of time , so that other professor may not perish of hunger .
* then, put both the left and right chopsticks down so that other may get a chance .
* repeat from the beginning.
* We Have

shared data : Spaghetti .

Process : Professors .

Resources Chopsticks .

**Deadlock** occurs if the resources are not enough that any of the professor is not able to eat , i,e when 1 stick is picked up by every process and no more sticks left as we know the process could not progress if they don’t have two chopsticks .

We solve this issue by using **Semaphores** that provides mutual exclusion for process , we have 1 semaphore for each process .

We have the array to represent the state for process , at any time process can either feel hungry , eat or think .

We made 4 functions namely :

**philosopher() :** Assign the job to each professor i,e picking the sticks and releasing the sticks .

- take\_Chopsticks()

- put\_ChopsticksDown()

**CheckPermission() :** Check the left and right side of the professor if free allow the process to take them and start eating .

1. **Project Result**

Understanding the real case scenario for deadlock , and solution for avoiding the deadlock .

1. **References**

Course Book