
Project Title:

# Dining Philosophers Problem

Submitted by:


| | |
|---|---|
| **Kashif Muneer** | **2021-CE-34** |
| **Talal Muzammal** | **2021-CE-47** |
| **Mehmood Ul Haq** | **2021-CE-35** |
| **Muhammad Shahzaib** | **2021-CE-41** |


Submitted to:

**Darakhshan Abdul Ghaffar**

Course:

**CMPE-331L Operating Systems A**

Semester:

**05**

Date of Submission:

**09-Nov-2023**

---


**Department of Computer Engineering**

**University of Engineering and Technology, Lahore**

# Table of Contents

## Abstract:

This research describes how to avoid deadlock condition in dining philosophers' problem. It is the undesirable condition of concurrent systems. It is marked as in a circular waiting state. At first, most people wear concepts of simple synchronization supported by the hardware, such as user or user interrupt routines that may have been implemented by hardware. In 1967, Dijkstra proposed a concept wearer an integer variable to count the number of processes that are active or who are inactive. This type of variable is called semaphore. The semaphore can also be used to synchronize the communication between devices in the device. In this journal, semaphore is used to solve the problem of synchronizing dining philosophers' problem. Dining itself is a situation where five philosophers are sitting at the dinner table to eat spaghetti, every philosopher is given a plate of spaghetti and one chopstick to eat spaghetti the two chopsticks are needed to resolve the issue. The semaphore variable is then applied to each chopstick chopsticks that can be shared by all the other philosophers. This paper presents the efficient distributed deadlock avoidance scheme using the lock and release method that prevents other threads in the chain from making race conditions.

## 1. Introduction:

The operating system is a program that links the user and the computer system. This operating system must be capable of controlling resource usage. In the process of designing the operating system, there is a common foundation called concurrency. Concurrent processes are when the processes work at the same time. This is called the multitasking operating system. Concurrent processes can be completely independent of each other but can also interact with each other. Processes that require synchronization to interact are properly controlled. However, in the concurrent processes that interact, there are some problems to be solved such as deadlock and synchronization.

## 2. Problem statement:

One of the classic problems that can illustrate the problem is the Dining Philosophers Problem. The Dining Philosopher's Problem can be illustrated as follows; there are five philosophers who would eat. On the table were reserved five chopsticks. If philosophers are really hungry, then it will take two chopsticks, which are in the right and left hands. However, sometimes only one course takes chopsticks. If there are philosophers who took two chopsticks, then there are philosophers who have to wait until the chopsticks are placed back. Inside this problem, there is the possibility of deadlock, a condition in which two or more processes cannot continue execution.

## 3. Proposed Methodology:

The basic idea behind the scenario is that if a concurrent activity always does what seems best for itself or what seems to be the right thing for itself in a shared resources scenario, the result

can be chaos. Is there a solution to the Dining Philosopher Problem? The scenario was posed not for a solution but to illustrate a basic problem if the traditional programming approach is applied to concurrent systems. The problem itself crops up in the concurrent systems, and the design decisions should be aware of this, and that is what we have to solve. Any set of concurrent programming techniques that we use is expected at the basic level to offer us features that can be used to deal with the Dining Philosophers' problem in some way.

There are several solutions which are: -

1. Allows at most four philosophers who sit together at one table.
2. Allows a philosopher to take chopsticks only if both chopsticks are there.
3. Allow only Odd numbered philosophers to pick up the right chopstick first and then the left, while even-numbered philosophers pick up the left chopstick first and then the right.

The first and second solution is not appropriate that's why we use the third solution in our project to solve dining philosophers' problem.

## 4. Expected outcome/results:

The expected outcome or results of solving the Dining Philosophers problem should satisfy the following criteria:

**Mutual Exclusion:** Only one philosopher can pick up both of their designated chopsticks and eat at any given time. This ensures that no two adjacent philosophers are eating simultaneously, maintaining mutual exclusion.

**No Deadlock:** The solution should prevent situations where all philosophers are waiting indefinitely for chopsticks, causing a deadlock. Deadlock is avoided by using a proper synchronization mechanism.

**No Starvation:** Starvation should be minimized, ensuring that all philosophers eventually get a chance to eat. Philosophers should not be unfairly denied the opportunity to eat

**Fairness:** The solution should aim for fairness, meaning that philosophers should take turns eating, and no philosopher should monopolize the resources.

## 5. Applications in Operating Systems:

Its applications in operating systems are primarily related to teaching and understanding key concepts and techniques for managing concurrent access to shared resources. Here are some of its applications in operating systems:

**Resource Allocation:** In an operating system, resources like CPU time, memory, and I/O devices are shared among multiple processes. The Dining Philosophers problem can serve as a metaphor for managing access to these resources to prevent conflicts, deadlocks, and resource starvation.

**Mutex and Semaphore Concepts:** The problem helps teach and reinforce the use of synchronization primitives like mutexes and semaphores to ensure exclusive access to shared resources. These concepts are fundamental in developing efficient and safe operating systems.

**Deadlock Detection and Prevention:** Operating systems need to detect and prevent deadlock situations where processes are waiting for resources that will never be released. The Dining Philosophers problem illustrates the importance of deadlock detection and prevention mechanisms in OS design.

**Scheduler Design:** Operating systems need to schedule processes or threads for execution. The Dining Philosophers problem can be used to discuss scheduling algorithms that ensure fairness and prevent resource starvation, similar to how philosophers take turns to eat.

# References


Valluri, C. (2022, November 13). The Dining Philosophers Problem Project. GitHub. https://github.com/chanakyav/The-Dining-Philosophers-Problem-Project

Anna, D. (2020, October 24). *The Dining Philosopher's Problem*. Medium.

https://medium.com/swlh/the-dining-philosophers-problem-bbdb92e6b788