ADITYA ENGINEERING COLLEGE (A)
Aditya Nagar, ADB Road, Surampalem
Department of Information Technology

Name of the Faculty: Mr. A.K.Chakravarthy, Assistant Professor, Department of IT
Subject: Operating Systems
Year & Semester: II-I
Topic: Dining philosophers' problem
Conventional Methods: Chalk & Talk
Teaching Methodology: Video Lecture

In Operating Systems, Dining philosophers' problem is an important topic in synchronization. Most of the teachers used conventional method of chalk & talk for delivering this topic. A new teaching methodology is required to better understand this topic.A "Video Lecture" innovative teaching method is used for teaching the Dining philosophers' problem through which students can completely understand indepth knowledge about solving Dining philosophers' problem.

References:

1. https://www.tutorialspoint.com/dining-philosophers-problem-dpp
2. https://www.studytonight.com/operating-system/dining-philosophers-problem
3. https://www.codingninjas.com/codestudio/library/dining-philosopher-problem-using-semaphores-2383

The dining philosopher's problem is the classical problem of synchronization which says that Five philosophers are sitting around a circular table and their job is to think and eat alternatively. A bowl of noodles is placed at the center of the table along with five chopsticks for each of the philosophers. To eat a philosopher needs both their right and a left chopstick. A philosopher can only eat if both immediate left and right chopsticks of the philosopher is available. In case if both immediate left and right chopsticks of the philosopher are not available then the philosopher puts down their (either left or right) chopstick and starts thinking again.

The dining philosopher demonstrates a large class of concurrency control problems hence it's a classic synchronization problem.

Five Philosophers sitting around the table

Dining Philosophers Problem- Let's understand the Dining Philosophers Problem with the below code, we have used fig 1 as a reference to make you understand the problem exactly. The five Philosophers are represented as P0, P1, P2, P3, and P4 and five chopsticks by C0, C1, C2, C3, and C4.

The solution of the Dining Philosophers Problem

We use a semaphore to represent a chopstick and this truly acts as a solution of the Dining Philosophers Problem. Wait and Signal operations will be used for the solution of the Dining Philosophers Problem, for picking a chopstick wait operation can be executed while for releasing a chopstick signal semaphore can be executed.

Semaphore: A semaphore is an integer variable in S, that apart from initialization is accessed by only two standard atomic operations - wait and signal. From the above definitions of wait, it is clear that if the value of S <= 0 then it will enter into an infinite loop(because of the semicolon; after while loop). Whereas the job of the signal is to increment the value of S.

Solution : Let value of i = 0(initial value), Suppose Philosopher P0 wants to eat, it will enter in Philosopher() function, and execute **Wait(take_chopstickC[i]);** by doing this it holds **C0 chopstick** and reduces semaphore C0 to 0**,** after that it execute **Wait(take_chopstickC[(i+1) % 5]);** by doing this it holds **C1 chopstick**(since i =0, therefore (0 + 1) % 5 = 1) and reduces semaphore C1 to 0.

Similarly, suppose now Philosopher P1 wants to eat, it will enter in Philosopher() function, and execute **Wait(take_chopstickC[i]);** by doing this it will try to hold **C1 chopstick** but will not be able to do that**,** since the value of semaphore C1 has already been set to 0 by philosopher P0, therefore it will enter into an infinite loop because of which philosopher P1 will not be able to pick chopstick C1 whereas if Philosopher P2 wants to eat, it will enter in Philosopher() function, and execute **Wait(take_chopstickC[i]);** by doing this it holds **C2 chopstick** and reduces semaphore C2 to 0, after that, it executes **Wait(take_chopstickC[(i+1) % 5]);** by doing this it holds **C3 chopstick**(since i =2, therefore (2 + 1) % 5 = 3) and reduces semaphore C3 to 0.

Hence the above code is providing a solution to the dining philosopher problem, A philosopher can only eat if both immediate left and right chopsticks of the philosopher are available else philosopher needs to wait. Also at one go two independent philosophers can eat simultaneously (i.e., philosopher **P0 and P2, P1 and P3 & P2 and P4** can eat simultaneously as all are the independent processes and they are following the above constraint of dining philosopher problem)

The drawback of the above solution of the dining philosopher problem

From the above solution of the dining philosopher problem, we have proved that no two neighboring philosophers can eat at the same point in time. The drawback of the above solution is that this solution can lead to a deadlock condition. This situation happens if all the philosophers pick their left chopstick at the same time, which leads to the condition of deadlock and none of the philosophers can eat.

Dining Philosophers Problem using Video Lecture:

Dining Philosophers Problem is demonstrated through video lecture. Video lecture is arranged by the faculty in the class to motivate students and make them understanding Dining Philosophers Problem concept easily.

https://www.youtube.com/watch?v=FYUi-u7UWgw