



The **d** **i** **n** **i** **n** **g**

**Philosopher problem**

**Alla Alwindawi**  
**2016-2017 spring term**  
**Istanbul Kemerburgaz university**

# What is Dining philosopher problem



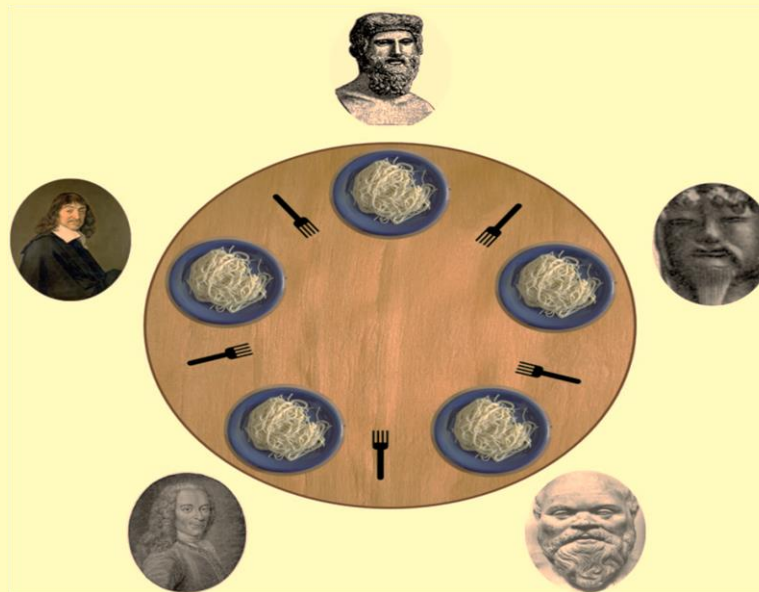
- In Computer science, the dining philosophers problem is an example problem often used in concurrent algorithm design to illustrate **Synchronization issues** and techniques for resolving them.
- It a theory was originally formulated in 1965 by Dutch computer scientist called **Edsger Dijkstra** .
- It's concerning resource allocation between processes. it is a model and universal method for testing and comparing theories on resource allocation.
- Dijkstra hoped to use it to help create a layered operating system.

# what is the problem about



It's talking about

- that five philosophers sit around table with five bowls of spaghetti
- A fork is placed Between each pair of bowls of spaghetti
- they spend their lives just thinking and eating.

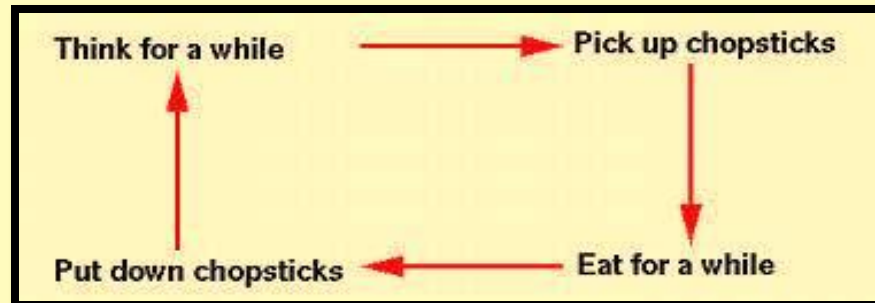


# The Rule

- The philosophers can't connect with each other
- A philosopher is either eating or thinking.
- When a philosopher wants to eat, he uses two forks - one from their left and one from their right.
- When a philosopher wants to think, he keeps down both forks at their original place.
- Philosopher can't start eating until he has both of them "2 forks".

## Analysis :

How do we write a threaded program to simulate philosophers?



How we can defined  
“**Thinking , Hungry and Eating** “ in the operating  
system ?



- **Thinking** : executing independently .
- **Hungry** : requesting a resource
- **Eating** : using shared resource.

# Problems

1. *The most serious problem of this program is that deadlock could occur!*

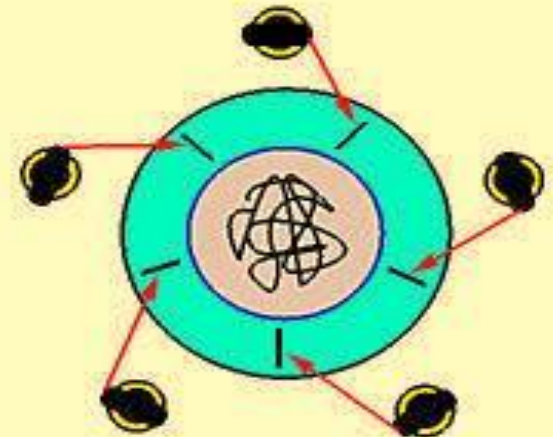
What if every philosopher sits down in the same time and picks up his left fork as shown in the figure?


In this case, all forks are locked and none of the philosopher can successfully lock his right fork .

As a result, we have a circular waiting

“(every philosopher waits for his right fork that is currently being locked by his right neighbor) “

and hence a deadlock occurs.





What is  
Deadlock??



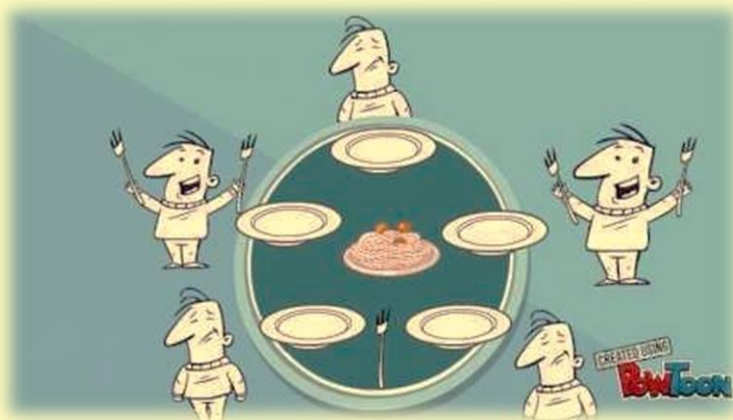
**A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs will stop to function.**



## 2. Starvation is also a problem!

- Imagine that two philosophers are fast thinkers and fast eaters. They think fast and get hungry fast.
- Then, they sit down in opposite chairs as shown below.
- Because they are so fast, it is possible that they can lock their forks and eat. After finish eating and before their neighbors can lock the forks and eat, they come back again and lock the forks and eat.
- In this case, the other three philosophers, even though they have been sitting for a long time, they have no chance to eat. This is a *starvation*.

“**Note** that it is not a deadlock because there is no circular waiting, and every one has a chance to eat!”







# Problems

- The problem was designed to illustrate the challenges of avoiding **Deadlock**- a system state in which no progress is possible
- How to design a **Concurrent Algorithm** such that each philosopher won't starve , he can forever continue to alternate between eating and thinking assuming that any philosopher cannot know when other may want to eat or think .

## Resource Hierarchy Solution

- Assigns a partial to the resources (fork)
- All resources will be requested in order and no 2 resources unrelated by order will ever be used by a single unit of work at the same time .
- Resources are numbered 1-5 and each philosopher will always pick up the lower-numbered fork and then the higher-numbered fork.
- If 4 of 5 philosopher together pick up their “lower fork” , only one “high fork “ will remain on the table .

So

- ❖ The 5<sup>th</sup> philosopher will NOT be able to pick up any fork .
- ❖ Also only 1 philosopher will have access to the “high fork” .

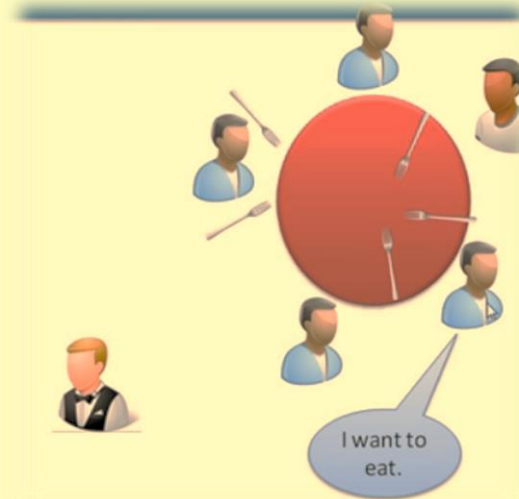
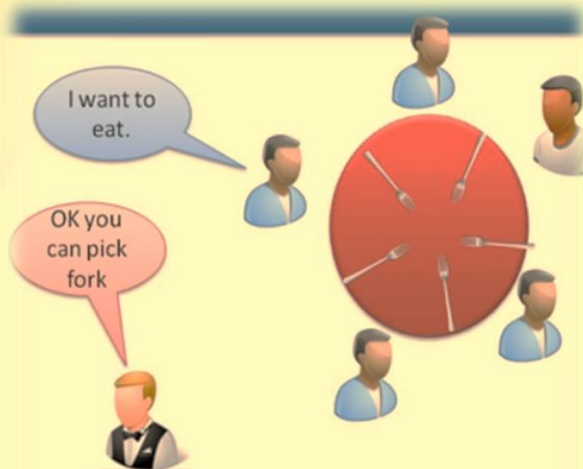


## Arbitrator solution

Guarantee that a philosopher can pick up only 2 solution by introducing an arbitrator Ex: Waiter

- In order to pick up the fork , a philosopher must ask waiter's permission
- Waiter gives permission to only 1 philosopher at a time until he has picked up both his fork
- Putting down a fork is always allowed.

\* Waiter is implemented as a *mutex* – a program object that allows multiple program threads to share same resource, such as file access , but not simultaneously .





## what are the advantages of this problem in operating systems?

- Almost all the current operating systems are systems support more than one task in the same time.
- That you can browse the Internet and use Word file at the same time.
- The operating system needs to support these programs (philosophers) through the memory and processing power of their distribution (forks) without access to state of the deadlock, and without disable one of the programs entirely.
- Some sources can not be used at the same time more than one program (one fork).
- therefore to make it easy to understand this problem and to solve it , Dijkstra has clarify it in this way



## Conclusion

- First, when multiple threads or processes access multiple resources exclusively, you should worry about the deadlock.
- Second, you should worry about starvation, and the only way to prevent starvation is to enforce all threads/processes for getting unblocked.
- Third, usually you should treat all threads equally, so that no single thread gets more resources than the others due to your synchronization protocol.

The left side of the slide features a vertical column of abstract, colorful brushstrokes. The colors transition from red and orange at the top to yellow and green at the bottom. The strokes are thick and expressive, with some overlapping and fading into the background.

**Thank you for  
your attention**