İstanbul
Bilgi Üniversitesi
LAUREATE INTERNATIONAL UNIVERSITIES

DINING PHILOSOPHERS PROBLEM

Özgür Özdemir
Anıl Kızıltan

CMPE 312
Final Project

1 History

Dining Philosophers Problem was acquired by Dijkstra in the literature and dining philosophers problem show us concurrent process management. 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.

2 Explanation

Generally, Dining Philosophers problem is drawbacks in operating systems. Actually, there a couple of problems in operating systems. Firstly, if there are philosophers in your operating system, it's probably not very efficient. Philosophers are not known for executing quickly. Quite the opposite in fact.And if you have forks in your CPU, then something is really, really wrong. Secondly, if you have mutex problems similar to the dining philosopher's problem, you typically have truly horrible performance problems and need to re-think your programs. And it describes a type of deadlocking problem which modern system are carefully designed to avoid.

Most popular example is that five silent philosophers sit a round table with bowls of spaghetti. Forks ara 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 forks. Each fork can be held by only one philosopher and so a philosopher can use the fork 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 forks. 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 behavior (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.

3 Solutions of Dining Philosophers

1. Semaphore Solution: There are three states of philosopher thinking, hungry and eating. There are two semaphores : Mutex and a semaphore array for the philosophers. Mutex is used such that no two philosophers may access the pickup or putdown at the same time. The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.

2. Random Solution: In this solution, it is envisaged that philosophers will act completely randomly to solve the problem. If philosophers can take second after they have taking a fork, they eat their food. If they do not take a second fence, they wait random amount of time and wait for the second fork to empty.If the other fork is left by the philosopher next to him during this period, he will pick up a crackle and eat his food. If the second fork is not released during this random period that he waits, then he forgets to eat the rest of the flared philosopher's food he holds in his hand.
   As noted in this solution, the process is entirely based on randomness. Accordingly, the system may not work properly. There is absolutely no guarantee that the system will work. The most important feature of solution that in this solution of code is easier that other system.

3. Conductor Solution: The waiter follow of the number of forks that are constantly empty at the table and used by philosophers to eat. In way, every philosophers have to ask waiter for taking fork. If waiter do not let, philosophers can not take fork in the table. In this solution, philosophers are prevented from encountering starvation because a logical waiter design allows all philosophers to eat. At the same time, the possibility of a deadly lock (deadlock) is resolved because the waiter will not wait forever for any philosopher. So the problem of the philosophers becoming aged forever waiting for each other is resolved.

4. Monitor Solution: This solution is similar conductor solution. The goal rank to each philosophers as a eating an non-eating. Each philosophers sort in specific order. Then if philosopher is eating the food by checking the state of the previous philosopher, philosopher does not eat . if he is not eating the previous philosopher himself.

5. Chandy Misra Solution: The most important feature of solution is that it removes a central decision-making mechanism
   This solution consist of 4 steps:

   1- For every pair of philosophers contending for a resource, create a fork and give it to the philosopher with the lower ID (*n* for agent P_n). Each fork can either be *dirty* or *clean.*Initially, all forks are dirty.

   2- When a philosopher wants to use a set of resources (*i.e.* eat), said philosopher must obtain the forks from their contending neighbors. For all such forks the philosopher does not have, they send a request message.

   3-When a philosopher with a fork receives a request message, they keep the fork if it is clean, but give it up when it is dirty. If the philosopher sends the fork over, they clean the fork before doing so.

   4- After a philosopher is done eating, all their forks become dirty. If another philosopher had previously requested one of the forks, the philosopher that has just finished eating cleans the fork and sends it. And this solution solve starvation problem. Philosophers rule not speaking each other but this solution developed that philosophers remove of not speaking each other.

4 Our Solution

The semaphore mechanism which is original treatment by Dijkstra is the one that we used for solution of Dining Philosopher Problem. In briefly, there is three states such that think, eat and hungry. And also a mutex and an array was used for such purposes; the mutex was used for not allowing to two process execute at the same time; the array was used for controlling each philosophers' behavior. The state mechanism work like that if the philosopher whose neighbors are not eating will take the forks and eat the spaghetti. After eat the spaghetti, he will put the forks down and go to status of thinking. However, there is some problems of this solution. One of them is deadlock. Let's assume that all philosopher in the table took right fork. It causes that there is no philosopher in the table who has got left fork, so it ended up with deadlock. Another problem is starvation. There is chance to one of the processes cannot pick forks because they are always busy by others. It will cause that this process will never execute. The possible solution of them can be putting another man who has authority to choose who will eat. He will give the forks to philosophers equally or least enough everyone to eat.

5 Build of Solution

The code of solution was written in C language. In order to build the code of our solution, there should be a computer which has C compiler.

To build the solution in terminal execute the code given below. Be sure you are in the correct path.

```
>> gcc solution.c –std=c99 –o solution
```

You will see that the code was compiled. Ignore the warnings if there are.

```
>> ls
 solution*   solution.c
```

Then you can simulate the solution by executing line given below.

```
>> ./solution
```

Here there are some example of simulation.

```
Philosopher # 2 is HUNGRY
Philosopher # 0 is HUNGRY
Philosopher # 1 is HUNGRY
Philosopher # 4 is HUNGRY
Philosopher # 3 is HUNGRY
Philosopher # 0 TAKING fork 4 and 0
Philosopher # 2 TAKING fork 1 and 2
Philosopher # 0 is EATING
Philosopher # 2 is EATING
```