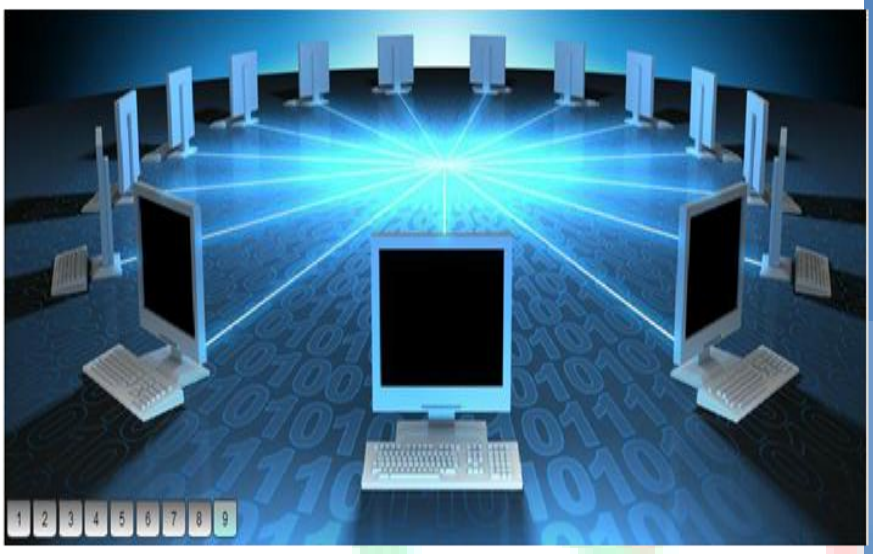
International Journal of Technology, Management & Humanities



www.ijtmh.com
ISSN (e) : 2454—566X

Volume - 1, Issue - 4
March - 2016

International Journal of Technology, Management and Humanities (IJTMH) refereed e-journal form in English.
International Journal of Technology, Management and Humanity is published on Quarterly basis with the aim to provide an appropriate platform presenting well considered, meaningful, constructively thought provoking and non-controversial but critically analyzing and synthesizing present and future aspects of Technical & scientific Education System with particular reference to the world.
The following types of article will be considered types of article will be considered

1. Research Articles: Original research in different fields of Science, Engineering and Management, Humanities will be evaluated as research articles.
2. Research Notes: These include articles such as manuscripts.
3. Reviews: Reviews of recent improvements, discoveries, developments, and thoughts in various fields of Science, management, and Engineering will be considered.
4. Frequency: FOUR issues in a year.

Indexing

JadounScience
http://www.jadounscience.com

International Society of Universal Research in Sciences

INDEX COPERNICUS INTERNATIONAL

DOAJ DIRECTORY OF OPEN ACCESS JOURNALS

Google Scholar

CABELL PUBLISHING, INC. Directories of Academic Journals

EBSCO HOST Electronic Journals Service

IJTMH

Contact Us
E-mail
submission@ijtmh.co
submissionijtmh@gmail.com
editor@ijtmh.com
editorijtmh@gmail.com

Analysis of the Dining Philosophers Problem in Concurrency Control between Processes

Author

[1] Dr. Zubair Ali, [2] Ashraf Ali

[1](Associate Professor / Department of Computer Science/Osmania University, Hyderabad)
[2] (Research Scholar/Department of Computer Science/ Osmania University, Hyderabad/India)

Abstract : *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.*

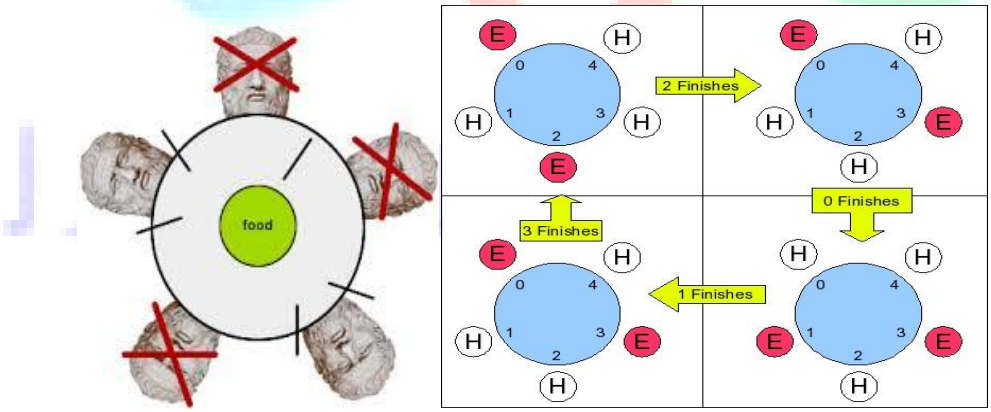
Keywords : *Mutex, Concurrency control, Fork, Resource Starvation*

1. Introduction

Five silent philosophers sit at a round table with bowls of spaghetti. Forks are placed between each pair of adjacent philosophers.

Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when he has 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 he finishes eating, he needs to put down both forks so they become available to others. A philosopher can take the fork on his right or the one on his left as they become available, but cannot start eating before getting both of them.
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.



2. Problems

The problem was designed to illustrate the challenges of avoiding 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 fork is available; when it is, pick it up;
- think until the right fork is available; when it is, pick it up;
- when both forks are held, eat for a fixed amount of time;
- then, put the right fork down;
- then, put the left fork down;
- repeat from the beginning.

This attempted solution fails because it allows the system to reach a deadlock state, in which no progress is possible. This is a state in which each philosopher has picked up the fork to the left, and is waiting for the fork to the right to become available. With the given instructions, this state can be reached, and when it is reached, the philosophers will eternally wait for each other to release a fork.

Resource starvation might also occur independently of deadlock if a particular philosopher is unable to acquire both forks because of a timing problem. For example there might be a rule that the *philosophers put down a fork after waiting ten minutes for the other fork* to become available and wait a further ten minutes before making their next attempt. This scheme eliminates the possibility of deadlock (the system can always advance to a different state) but still suffers from the problem of livelock. *If all five philosophers appear in the dining room at exactly the same time and each picks up the left fork at the same time the philosophers will wait ten minutes until they all put their forks down and then wait a further ten minutes before they all pick them up again.*

Mutual exclusion is the basic idea of the problem; the dining philosophers create a generic and abstract scenario useful for explaining issues of this type. The failures these philosophers may experience are analogous to the difficulties that arise in real computer programming when multiple programs need exclusive access to shared resources. These issues are studied in the branch of concurrent programming. The original problems of Dijkstra were related to external devices like tape drives. However, the difficulties exemplified by the dining philosophers problem arise far more often when multiple processes access sets of data that are being updated. Systems such as operating system kernels use thousands of locks and synchronizations that require strict adherence to methods and protocols if such problems as deadlock, starvation, or data corruption are to be avoided.

3. Solutions

Basically there are three solutions for the above problem

3.1. Resource hierarchy solution

This solution to the problem is the one originally proposed by Dijkstra. It assigns a partial order to the resources (the forks, in this case), and establishes the convention that all resources will be requested in order, and that no two resources unrelated by order will ever be used by a single unit of work at the same time. Here, the resources (forks) will be numbered 1 through 5 and each unit of work (philosopher) will always pick up the lower-numbered fork first, and then the higher-numbered fork, from among the two forks he plans to use. ***The order in which each philosopher puts down the forks does not matter*.** In this case, if four of the five philosophers simultaneously pick up their lower-numbered fork, only the highest numbered fork will remain on the table, so the fifth philosopher will not be able to pick up any fork.

Moreover, ***only one philosopher will have access to that highest-numbered fork, so he will be able to eat using two forks.***

While the resource hierarchy solution avoids deadlocks, it is not always practical, especially when the list of required resources is not completely known in advance. For example, if a unit of work holds resources 3 and 5 and then determines it needs resource 2, it must release 5, then 3 before acquiring 2, and then it must re-acquire 2 and 3 in that order. Computer programs that access large numbers of database records would not run efficiently if they were required to release all higher-numbered records before accessing a new record***, making the method impractical for that purpose.***

3.2. Arbitrator solution

Another approach is to guarantee that a philosopher can only pick up both forks or none by introducing an arbitrator, e.g., a waiter. In order to pick up the forks, a philosopher must ask permission of the waiter. ***The waiter gives permission to only one philosopher at a time until he has picked up both his forks. Putting down a fork is always allowed.*** The waiter can be implemented as a ***mutex***. In addition to introducing a new central entity (the waiter), this approach can result in reduced parallelism: if a philosopher is eating and one of his neighbours is requesting the forks, all other philosophers must wait until this request has been ***fulfilled even if forks for them are still available.***

3.3. Chandy/Misra solution

In 1984, K. Mani Chandy and J. Misra[5] proposed a different solution to the dining philosophers problem to allow for arbitrary agents (numbered P1, ..., Pn) to contend for an arbitrary number of resources, unlike Dijkstra's solution. It is also completely distributed and requires no central authority after initialization. However, it violates the requirement that "the philosophers do not speak to each other" (due to the request messages).

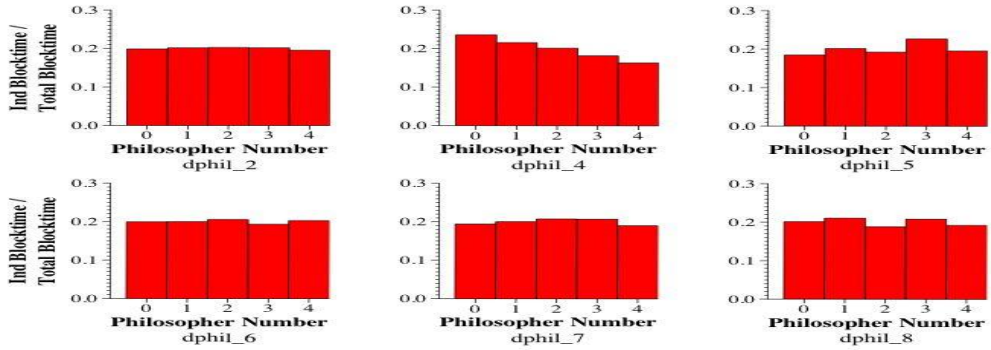
1. For every pair of philosophers contending for a resource, create a fork and give it to the philosopher with the lower ID (n for agent Pn). Each fork can either be dirty orclean. Initially, all forks are dirty

2. When a philosopher wants to use a set of resources (i.e. eat), he must obtain the forks from his contending neighbours. For all such forks he does not have, he sends a request message.

3. When a philosopher with a fork receives a request message, he keeps the fork if it is clean, but gives it up when it is dirty. If he sends the fork over, he cleans the fork before doing so.

4. After a philosopher is done eating, all his forks become dirty. If another philosopher had previously requested one of the forks, he cleans the fork and sends it.

4. Result: In their analysis they derive a system of preference levels from the distribution of the forks and their ***clean/dirty states***. They show that this system may describe an acyclic graph, and if so, the operations in their protocol cannot turn that graph into a cyclic one. This guarantees that deadlock cannot occur. However***, if the system is initialized to a perfectly symmetric state, like all philosophers holding their left side forks, then the graph is cyclic at the outset, and their solution cannot prevent a deadlock***. Initializing the system so that philosophers with lower IDs have dirty forks ensures the graph is initially acyclic.



5. Conclusion:

This solution also allows for a large degree of concurrency, and will solve an arbitrarily large problem.

It also solves the starvation problem. The clean / dirty labels act as a way of giving preference to the most "starved" processes, and a disadvantage to processes that have just "eaten". One could compare their solution to one where philosophers are not allowed to eat twice in a row without letting others use the forks in between. Their solution is more flexible than that, but has an element tending in that direction.

Refrences

1. *Silberschatz, Abraham; Peterson, James L. (1988). Operating Systems Concepts. Addison-Wesley. ISBN 0-201-18760-4.*
2. *Dijkstra, E. W. (1971, June). Hierarchical ordering of sequential processes. Acta Informatica 1(2): 115–138.*
3. *Lehmann, D. J., Rabin M. O, (1981). On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem. Principles Of Programming Languages 1981 (POPL'81), pp. 133–138.*