Grenze International Journal of Engineering and Technology, Special Issue

An Algorithmic Approach for Deadlock Detection in Hypervisor

Madham Naren[1] and Sandeep Ravikanti[2]
[1]Methodist College of Engineering and Technology, Hyderabad, India
Email: narenmadham310@gmail.com
[2]Assistant Professor, CSE Dept, Methodist College of Engineering &Technology, Abids,Hyderabad, India
rsandeep@methodist.edu.in

***Abstract*—A Deadlock is a state in which a process waits for the resource that is being held by another process. The processes will be blocked completely and it doesn't lead to any progress. Many of the present algorithms take polynomial time only to detect the deadlock. The proposed algorithm provides an approach to detect and recover from the deadlock in linear time complexity. The algorithm shows a one-to-one correspondence to the figures formed using the Penrose Geometry. The algorithm can detect the deadlock without ambiguity even if the resources with multiple instances are involved.**

***Index Terms*— Deadlock, Penrose Geometry, Linear time, Unambiguous, Multiple instances.**

I. INTRODUCTION

The deadlock is a state of the system in which the processes waits for the completion of another process which may never completes i.e. the time taken for the process to wait is indefinite. All the existing algorithms are confined only to the deadlock detection or deadlock prevention. In this algorithm, we are going deal with both detecting the deadlock in any system if exists[2] and preventing the system from deadlock.

It is most common to see that the server of the websites may be down for a specific amount of time. This might be because of the deadlock that occurs in the hypervisor, due to improper resource allocation. The Paas service of the Cloud computing, provides an opportunity to implement the website using different operating systems. Since the servers will be running in the different operating systems, they will be unaware of the systems. The Hypervisor must handle have to handle the deadlock.

In the most of the initial operating systems, the deadlock is ignored. At few circumstances, the system may have more number of processes requesting for the resources whereas as the CPU does not have sufficient resources to provide. In these situations, processes continue to be in the waiting state and the system needs to need to be rebooted which may lead to the deletion of the saved work of other processes.

In the present Operating systems[3] the deadlock is considered. These systems uses the algorithms such as Banker's algorithm and the Resource Allocation Graph that take polynomial time to solve the problem as they make use of matrices and works only for the resources with single instance. Few algorithms take multiple cores of CPU[1] to solve the process that degrades the CPU performance.

Grenze ID: 01.GIJET.4.3.14

The algorithm proposed uses a different approach to solve the problem. The CPU will make use of only one process to solve the problem. This algorithm doesn't restrict to the number of instances of the resource. Here, we make use of figures generated using the Penrose Geometry[8] to understand the algorithm as the structure of this figure is much related to the algorithm being proposed. The time taken by the algorithm is linear which is much faster when compared to the existing solutions[7].

A. Related Work

The most famous algorithm used for detecting the deadlock is the Banker's Algorithm. The algorithm considers many number of matrices, which leads in the traversal of both row-wise and column-wise iterations. Therefore, the algorithm takes $\theta(n^2m)$ i.e. polynomial time to detect the deadlock. Another approach for detecting the deadlock is the 'Resource Allocation Graph'[1][5]. The algorithm is to construct a graph the processes and the resources as the nodes of the graph. The system is said to be in the deadlock if any cycle exists in the graph. The algorithm works only with the resources having a single instance. The output may be ambiguous. In 2015, Ha Huy Cuong Nguyen,[2] had proposed an algorithm to detect the deadlock. The algorithm will take the Resource allocation graph into consideration. Then, the path matrix is constructed from the graph and the cycles are deadlock. The algorithm works only for the resources with single instance. Concept of Parallelism is used to run the algorithm in linear time [4].

B. Abbreviations and Acronyms

PaaS is an abbreviation of one of the service of the cloud computing, Platform as a Service. It means that the hypervisor[5] can allow multiple servers to run on the different operating systems.

C. Equations

For a system of n processes, having m instances of the resources available, the necessary condition that is to be satisfied to avoid the deadlock is

$$\sum P_i <= m + n$$

Where, '$\sum$' means summation. P_i denotes the request sent by the process. Therefore, $\sum Pi$ gives the total count of all the process requests.

II. PENROSE GEOMETRY

The Penrose geometry can be defined as a way of representing the relation between the processes involved in the deadlock. The Penrose geometry is practically impossible. The Penrose shapes are used only for the sake of understanding the property of cycles[8]. Hence, they are not considered as the data structures. In terms of the Penrose Geometry, the beams will have a specific shape of a simple closed figure, i.e. either a triangle or a square and so on. All the processes of the system are represented as a single beam in the Penrose shape. The number of surfaces of the Penrose triangle depends on the shape of the beam considered and the number of processes participating. The control of the CPU is referred to as the ball that gets stuck inside the Penrose figure. If a ball goes through all the beams of the triangle, it visits all the surfaces only once. This is the major advantage of the Penrose geometry. It becomes complex to imagine a figure when the number of the surfaces processes increases more than 7[9].

The Penrose geometry can be extended up to many edges, taking different shapes of beams into consideration. The Penrose geometry is more likely to be visualized only if we use a quadrilateral beam and 3 edges as shown below[3].

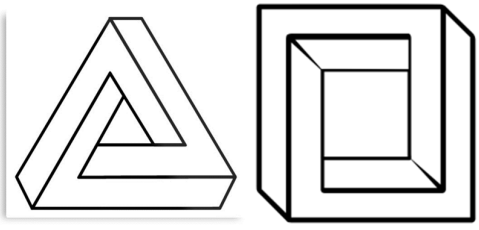

Fig 2.1: Penrose Triangle And Penrose Square

III. ACTUAL ALGORITHM

Input:
The state of the system with the number of processes, resources and their corresponding requests to the resources.

Output:
Declare if the given system is in Deadlock or not, without ambiguity.

Necessary Condition:
The sum of all the requests of the processes must be less than the sum of available resources and the number of processes.

$$\sum P_i <= m + n$$

Where,

$\sum Pi$ = the count of total process requests
m = the total number of resource instances
n = the total number of the processes.

If the necessary condition is violated, the algorithm straight away declares that the system is in deadlock.

Algorithm:

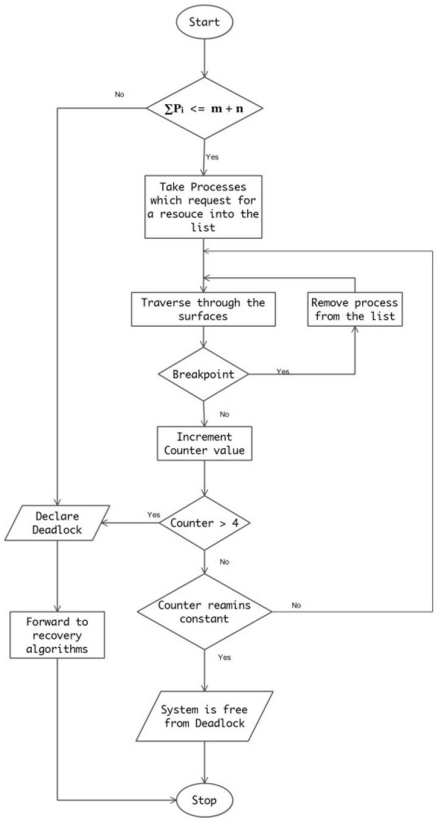

Fig. 3.1 Flowchart of the algorithm

A. Execution of the Algorithm

Consider the above snapshot of the system, represented in the form of a Resource Allocation graph. All the rectangles of form 'Pi' are the processes. All the ellipses in the form 'Qi' are the resources. All the resources can have any number of instances. Assume that the Penrose figure[4] is constructed using a quadrilateral beam.

Phase 1: Checking for the necessary condition:

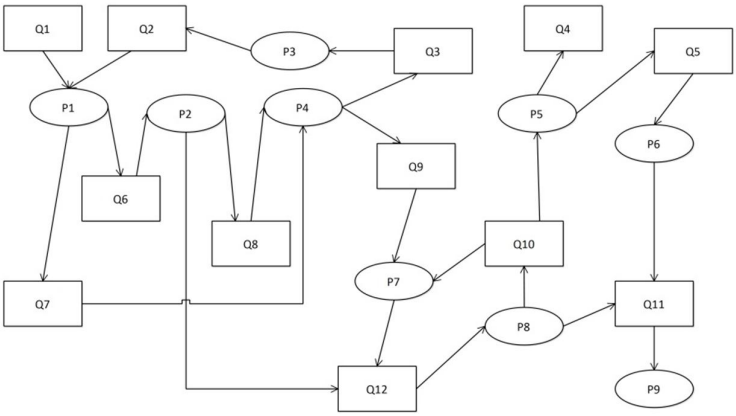

Fig 3.2: Snapshot of the System

Here, the number of process requests is greater than the summation of the number of processes and the number of instances of the resources. Hence, we can proceed to the algorithm.

$$\sum Pi <= m + n$$

Phase 2: Construction of Penrose figure:

Here, since the process 'p9' is not requesting for any resource, we can ignore 'p9'. The free instance of the resource is allotted to p6. Hence, ignore p6. The free resource is allotted to p5. Hence ignore p5. Now, the key value pairs are given below.

{ p1: p2, p1:p4, p2: p4, p2: p8, p3: p1, p4: p7, p7: p8 }

Phase 3: Removing the breakpoints:

Consider all the processes into a list or an array. A counter value is assigned to all the processes. The value of the counter is incremented for every time we visit the process. Taking any process as the initial state, we traverse through all the processes of the Penrose figure. If we come across any such breakpoint[10], we remove that process from the array, indicating that the process cannot be involved in the deadlock. The same is repeated with all the processes. Here, we don't have any breakpoint, since all the possible breakpoints are removed.

Phase 4: Checking for deadlock:

If the array is empty, it indicates that all the processes are the breakpoints i.e. no process is involved in the deadlock. Hence we can declare that the system is not in deadlock. If the array is not empty, then at least one of the processes has its counter greater than 'four' (since, quadrilateral beam is used), concluding that the system is in deadlock. Here, we have the processes p1, p2, p3, p4, p7, p8 in the array. When we traverse from the process p1, we reach p4. Again, when we traverse from the process p4, we reach to the process p7[3]. From p7, we traverse to p8. From p8, we traverse to p3. From p3, we again traverse to p1 i.e. the point where the traversal started. After 4 consecutive repetitions, the value of the counter will be '4'. Hence we can declare that the system is in deadlock[6]. The system in now advanced to perform deadlock recovery strategies[9].

Proof for time complexity

Consider a situation in which every process of the system requests for every instance of the system of n processes and m resources. Then the process 1 requests for 'm' resources, process 2 requests for the same 'm' resources … and so on process 'n' requests for the same 'm' resources. Then,

For all (n,m) ϵ Natural numbers, such that $(n,m) \neq (2,2)$

$$(n * m) > (m + n)$$

Hence the necessary condition of the system is violated. Therefore the maximum summation of the processes cannot be more than (m+n-1). Therefore, the time complexity of the algorithm will be definitely less than $\theta(mn)$.

Therefore, the time complexity of the system is $\theta(n)$.

The worst case of the algorithm is when the system has (m+n-1) process requests.

IV. Conclusions

The deadlock detection that was assumed to take polynomial time is now reduced to a linear time complexity. Unlike the Resource Allocation Graph, the algorithm considers only processes during its execution, hence avoids traversing through all the resources again. Unlike Banker's algorithm, the matrices are avoided in the algorithm there by avoiding the polynomial iterations. The algorithm is not divided into threads. Hence it uses only a single processor core to execute and avoids the parallelism. Thus, the CPU performance will not be degraded. In future, the Algorithm can be used to detect the busy waiting condition expected to occur during the process synchronization. It helps in reducing the wastage of the CPU cycles. The concepts of Neural Networks and Machine Learning can be involved to check how frequently a resource is being used to make the algorithm efficient.

Acknowledgment

My Sincere thanks to Ha Huy Cuing Nguyen, Da Nang of University for clarifying my doubts that I had during the construction of the algorithm.

References

[1] Algorithmic approach to deadlock detection for resource allocation in heterogeneous platforms - HA HUY CUONG NGUYEN - Advances in Digital Technologies (March 2016)
[2] Energy Efficient Resource Allocation for Virtual Services Based on Heterogeneous Shared Hosting Platforms in Cloud Computing Nguyen Minh Nhut Pham, Van Son Le, Ha Huy Cuong Nguyen – (October 2015)
[3] A technical solution for resource allocation in heterogeneous distributed platforms Ha Huy Cuong Nguyen and Van Son Le – IJCSNS International Journal of Computer Science and Network Security, VOL.16 No.1, January 2016
[4] A New Technical Solution Prevention Deadlock for Resource Allocation in Heterogeneous Distributed Platforms - IJCSN International Journal of Computer Science and Network, Volume 4, Issue 2, April 2015
[5] S. Venkatesan, Tony Tong-Ying Juang, and Sridhar Alagar. 1997. Optimistic Crash Recovery without Changing Application Messages. IEEE Trans. Parallel Distrib. Syst. 8, 3 (March 1997), 263-271.
[6] Shokripour A., Othman M., Ibrahim H., Subramaniam S. (2011) A Method for Scheduling Heterogeneous Multi-Installment Systems. In: Nguyen N.T., Kim CG., Janiak A. (eds) Intelligent Information and Database Systems. ACIIDS 2011. Lecture Notes in Computer Science, vol 6592. Springer, Berlin, Heidelberg
[7] Lim J., Suh T., Yu H. (2013) A Deadlock Detection Algorithm Using Gossip in Cloud Computing Environments. In: Han YH., Park DS., Jia W., Yeo SS. (eds) Ubiquitous Information Technologies and Applications. Lecture Notes in Electrical Engineering, vol 214. Springer, Dordrecht
[8] (2011) Deadlock Detection. In: Padua D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA
[9] Ashfield B., Deugo D., Oppacher F., White T. (2002) Distributed Deadlock Detection in Mobile Agent Systems. In: Hendtlass T., Ali M. (eds) Developments in Applied Artificial Intelligence. IEA/AIE 2002. Lecture Notes in Computer Science, vol 2358. Springer, Berlin, Heidelberg
[10] Locking and Deadlock Detection in Distributed Data Bases Daniel A. Menasce And Richard R. Muntz, Member, IEEE - IEEE transactions on software engineering, vol. se-5, no. 3, may 1979