A Survey of Concurrency Control Algorithms in the Operating Systems

Hossein Maghsoudloo
Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran

Rahil Hosseini
Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran (Corresponding Author)
rahil.hosseini@qodsiau.ac.ir

ABSTRACT

Concurrency control is one of the important problems in operation systems. Various studies have been reported to present different algorithms to address this problem, although a few attempts have been made to represent an overall view of the characteristics of these algorithms and comparison of their capabilities to each other. This paper presents a survey of the current methods for controlling concurrency in operating systems. Classification of current algorithms in operating systems has been proposed. Current concurrency control algorithms are classified into four groups: 1) software-based algorithms, 2) hardware-based algorithms, 3) based operating system, and 4) based on message passing. Furthermore, it presents an analysis of the capabilities and characteristics of current algorithms' in their own category (intra-group comparison analysis) and between different categories (inter-group comparison analysis) to put a light on the way of selecting a proper algorithm for various circumstances in operating systems.

Keywords

Concurrency control, Operating systems, Semaphore, Monitors

1. Introduction

In most modern systems, various numbers of processes concurrently cooperates with each other, as a whole software system. In other word, some of parallel processes run on many or even one a processor. In some of systems such as multi-programming systems, multi-processing, network operating systems, distributed operating systems and distributed systems, concurrency of processes is an attribute of the system. Parallel processes and concurrency of processes are discussed in three different areas, which are summarized as below [1]:

- **Multiple applications:** multi-programming can manage the time of process in sort of dynamically division between numbers of applied programs.
- **Structured applications:** for development and design principles and structured programming, a set of concurrent processes can be implemented.
- **Operating system structure:** through concurrency in operating sytems, identical rights are given to system's programs which can be implemented like a set of processes.

While parallel and concurrent performance of processes can boost system's efficiency, it can also cause some problems. Firstly, how processes can exchange information with each other? Secondly, what would be the solution for coping with a competitive situation. To address this issue, a simple solution is that , processes do not interfere with each other in reference sharing and competition for critical activities. Thirdly, when there is dependency between two or many processes, appropriate ranking of activity accomplishment or process synchronizing plays an important role.

Regarding the first problem, which is message exchanging, there are various solutions like duct, message exchanging and file sharing, but in all of these cases competency is significantly decreased because of system calling and also core's time-consuming trap. Fr this, this processes can share storage places to have closed and competent interactions to let all of them to write or read common data. This common place can be somewhere in the main memory . Concurrent processes can access to a common memory, at this point in order to avoid any problem in concurrent programs and access to critical resources is the same common memory. To address the second problem that is competitive situations, proposed is required to satisfy five conditions:

- **Mutually exclusive condition:** among all processes which having the critical zone for the same source or any mutual competitive factor, there is only one authorized process at the time which can enter the critical zone.
- **Promotion condition:** process, which does not intend to enter the critical zones currently, does not need the prohibition right for other processes entrance to critical zone.
- **Limit expectancy condition:** processes which, need to enter critical zone must have a limited awaiting time and do not encounter famine or even impasse.
- **Commonness condition:** proposed solution must not have any limitation on the processes' relative speed, their number and processor speed or number of processors.
- **Certain condition:** solutions do not have uncertain or random status.

2. Confronting solutions for competitive situations in concurrency of processes

Currently, competitive situations in processes are controlled using five groups of solutions, which are explained in the rest of this section.

2.1. Software-based solutions

In software-based solution, the burden of mutually exclusive condition satisfaction is in the charge of processes and written codes by programmer and there is no support from programming language and operating system. The only support is provided by hardware is just possible in a moment of accessing to one place of memory. In all of software algorithms, busy waiting problem exists which partly deteriorate this solution's competency. Some of the important software-based solutions presented in the literatures are as follows:

- Dekker's first method (strict alternation) [5]: it is double processes and has the problem of promotion condition disregarding. This speed of this algorithm is low because of decussate accessing to critical zone.
- Dekker's second method [7]: it is double processes and disregards the mutually exclusive condition and, therefore can cause starvation.
- Dekker's third method [7]: it is double processes and this method can lead to deadlock.
- Deker's forth method [7]: it is a double processes. In this solution there is problem named live lock which can be reminisce as an open-able dead-end, meanwhile this algorithm also has the problem of starvation.
- Dekker's final [5]: it is double processes and is the last algorithm which was proposed by Dekker. One of its biggest deficiency is its complexity.
- Peterson [4]: it is a double processes but the algorithm's complexity is less than the Dekker algorithms.
- Peterson functional call [4]: it is a double processes and is accomplished by functions which are in connection with each other by their parameters. It includes all Peterson's algorithm characteristic.
- Bakery [6]: this algorithm is like the usual intermittent method in bakery. This solution is used for indefinite number of processes, but does not guaranty that the number, which goes to processes as an intermittent be the same.

The crucial point here is that all of software solutions which are innovates as a double process can be extended to indefinite number of processes. Software solutions merely used for mutually exclusive control and not used in processes synchronizing [7].

2.2. Hardware-based solutions

In this method, in the design of the processor there are instructions for performing two reading and writing functions of one place in memory like atomic and a reactive cycle. While running these instructions, pause cannot do the text substitution and also lock of the memory gateway [1]. Scrutinized hardware solutions in this article consist of:

- **Deactivate pause instruction** [1]: mutual exclusive solution and limit expectation condition (cause of starvation) disregarded in this solution. The pause deactivation is useful in general but if we give the deactivation permission control to the user the we might face with the problem like mutual exclusive disregarding and starvation.
- **TSL instruction [3]**: this solution has the same common problem in software solution that is busy waiting, in addition to the starvation problem.
- **SWAP instruction [2]:** This solution suffers from the problem of busy waiting and starvation.

2.3. Operating systems-based solutions

Semaphores are the solution accomplished by operating systems. Three basic functions are defined on semaphores which using them the problem of mutual exclusive and synchronization of the processes are controlled [3]:

- Give some primary positive
- WAIT action, which reduced a unit and if it is negative, process will be in wait queue.
- SIGNAL action, which added a unit and if it was not positive, will release a barred process in queue.

Semaphores exist in two forms: general or binary, with the same capabilities, i.e., all issues, which are solvable by general semaphore can be solved by binary semaphores. For collecting processes, semaphores use a queue that wether this queue is FIFO kind, Semaphores acted fairly and idiomatically these semaphores named strong semaphores, and the semaphores, which have no specific policy on processes queue are called weak semaphores.

2.4. Programming language-based solutions

These types of solutions are implemented as monitors by programming language designers. A monitor consists of the sets of schemes, variations, data structures that are grouped with each other in a pack [8]. Monitors have two crucial characters:

- A process can run the schemes in the monitor whenever it want but cannot access to the monitor's inside data by monitors outside schemes that this feature is insulation.

- Every moment, just one process can be activated in monitor.

2.5. Messages

Messages are a mechanism to control synchronization and correlation between processes. The most important characteristic of messages is that they can be used in distributed system; messages accomplished by two calling systems SEND and RECEIVE.

As mentioned up to here, each one of these solutions has their advantages and disadvantages and even they are different in function range just as an example messages beside concurrent control on multiprocessing systems or multiprocessor they are easily capable of using in server systems [2].

3. Comparison of Current Algorithms for Concurrency Control in Operating Systems

Most of current solutions use mutual exclusive of the processes when dealing with the critical sections. However, the solutions based on operating systems are capable to control process synchronization. Therefore, concurrency control solutions are different in terms of their strength, number of issues covering and their complexity.

Table 1.1 brings a comparison of the current algorithms to cope with the problem of concurrency control. The comparison analysis is conducted in two different studies as pointed bellows:

1) **First study (Intra- Group):** comparison of the algorithms in their categories (e.g., comparison of the software based algorithms with each other)
2) **Second study (Inter-Group):** overall comparison of selected algorithms in the first study (various categories with each other)

The rest of this section presents the details and result of these two comparison analysis studies.

3.1 The First Study: Intra- Group Comparison Analysis

The concurrency control algorithms in operating systems are considered and classified according to the criteria of mutual exclusive conditions, promotion, limit waiting and busy waiting. Table 1 summarizes the concurrency control algorithms in operating systems in mutual exclusive conditions regarding promotion, limit waiting and busy waiting criteria.

3.1.1 Software-based algorithms

As shown in Table 1, software based algorithms are presented in [4]-[7]. All of software based algorithms have busy waiting problem in their nature which decreases the software algorithm competency's bonus.

Strict alternation algorithm disregards the promotion condition and even in the case of an empty critical zone while processes might not have the entering permission. Dekker's second algorithm does not accomplish mutual exclusive condition which is one of the important competitive conditions between processes. Furthermore, for the possibility of consecutive entering of a process to critical zone, may lead to starvation. Dekker's third algorithm has the dead end problem and two processes can wait up to infinity for entering to critical zone. Dekker's forth algorithm besides having starvation problem has the live lock problem that means none of these processes can enter the critical zone, although with change in relative speed of two processes this cycle can be broken. Bakery algorithm does not guarantee that the number which goes to processes as a turn be unique and can be used in a case that the number of two or more processes from FIFO queue be the same This algorithm's major problems are complexity and calculation, and intermittent system and intermittent control condition which reduces its competency.

Table 1: Intra-Group Comparison Analysis of Concurrency Algorithms

No.	Algorithm Name	Mutual Exclusive	Promotion Condition	Limit Waiting	Busy Waiting
1	Deactivating pause [1]	No	No	Starvation	Yes
2	SWAP [2]	Yes	Yes	Starvation	Yes
3	TSL [3]	Yes	Yes	Starvation	Yes
4	General semaphore [3]	Yes	Yes	Yes	No
5	Messages [3]	Yes	Yes	Yes	No
6	Monitor [3]	Yes	Yes	Yes	No
7	Peterson [4]	Yes	Yes	Yes	Yes
8	Peterson functional call [4]	Yes	Yes	Yes	Yes
9	binary semaphore [4]	Yes	Yes	Yes	No
10	Dekker's final [5]	Yes	Yes	Yes	Yes
11	Dekker's first [5]	Yes	No	Yes	Yes
12	Bakery [6]	Yes	Yes	Yes	Yes
13	Dekker's second [7]	No	Yes	Starvation	Yes
14	Dekker's third [7]	Yes	Yes	Dead lock	Yes
15	Dekker's forth [7]	Yes	Yes	starvation	Yes

Based on the Peterson's algorithm in Table 1, Peterson like call function, Dekker's five algorithm has all essential conditions for confronting with competitive condition. Peterson's algorithm has the lowest complexity between these three algorithms and it is easily possible to be accomplished. Therefore, the Peterson's algorithm is chosen for the second study of comparison analysis because of competent use of mutual resources, simplification and its comprehensiveness.

3.1.2 Hardware-based solutions

Hardware based solutions include deactivating pauses solution order, the TSL and the SWAP instructions. As shown in Table 1, all hardware based solutions have the problem of starvation and busy waiting. Between these three, the TSL and the SWAP algorithms are better because deactivating pause solution is just valid in mono processor and just deactivates that processor pause and other processor by continuously running pauses and turning off the pauses by user's process which threat security of the system. On the other side, the TSL and the SWAP instructions suffer from serious issues such as processor support of the TSL and the SWAP instructions. Therefore, according to the comparison of the hardware based solutions the TSL and SWAP solutions are selected and used in the second study of the comparison analysis.

3.1.3 Operating system based solutions (semaphores)

Semaphores are used for mutual exclusion regarding and synchronizing while former solution is only used for mutual exclusion. This can be count as a benefit for semaphores besides semaphores might have some problems in their functions. However, it was reported in some literatures that the semaphores may lead to dead lock but semaphore's do not have dead lock in their nature. This fact depends to how implement them in a programming language.

One of the advantages of semaphore is multi-processing, they do not have busy waiting and reverse preference issues which emphasizes on this type of algorithms' strength. Among the general and binary semaphores, binary semaphores were chosen for the second round of comparison. Using semaphores degrades the complexity

3.1.4 Programming language based solutions (monitors)

Using this method in general is easier than semaphores. Implementation of monitors is not too complicated. Furthermore, synchronization and mutual exclusion, and insulation are managed and the program is more isolated and can degrade the error percentage. The problem of some of conventional programming languages is that they do not support semaphores (such as C, and Pascal). Another important problem is that monitors are usable just in single processor systems or with the multi-processors with mutual memory.

3.1.5 Message passing solutions

Messages are more complicated mechanism than other solutions. This complication requires lots of adjustment to an algorithm to provide proper function such as bellows:

- Reliability and not losing sender's and receiver's messages
- Processes addressing
- Possessing sender's and receiver's identity for boosting system security
- System competency when sender and receiver are on one system.

This method is more effective for distributed system. This solution is more comprehensive and has similar competency on multi-programming, multi-processing, network operating systems, distributed operating systems, and distributed systems. However, messages are proper for distributed systems and distributed operating systems. Using them on other systems may deteriorate complexity and boosting error ratio. For this reason, they are not considered in second comparison study.

3.2 The Second Study: Inter-Group Comparison

In the first study, as summarized in Table 1, solutions are compared with each other in terms of mutual exclusive condition, promotion, limit waiting and busy waiting and the algorithms chosen in each category are as bellows:

- From software solution of Peterson's algorithm, because of competent use of mutual resources, simplification, comprehensiveness and regarding mutual exclusion condition, limit waiting and promotion.
- From hardware solution the TSL and the SWAP algorithm, because of considering mutual exclusion condition and promotion
- From operating systems solutions, binary semaphore were chosen because of mutual exclusion condition regarding, limit waiting, promotion, not having busy waiting and simplified accomplishment
- From programming languages solutions, monitors were selected because of their mutual exclusion condition regarding, limit waiting, promotion, not having busy waiting, insulation and reducing error percentage.

Algorithms selected for the second comparison analysis all consider the mutual exclusion condition and promotion. In the second comparison analysis, as shown in Table 2, solutions are considered and compared in terms of limit waiting condition, busy waiting and usage limitation conditions.

According to Table 2, Peterson algorithm has busy waiting problem. This algorithm besides having busy waiting problem has reverse preference problem (processes locking). Another point is that nature of this algorithm is binary but it can be extended to infinite number of processes which may increase complexity of the solutions.

The TSL and the SWAP do not consider the limit waiting condition and they may lead to starvation, besides having busy waiting and they can be used in a case which the processor support them.

Table 2: Inter Group Comparison Analysis of the Concurrency Algorithms

No.	Algorithm	Limit Waiting	Busy Waiting	Usage Limitation
1	Peterson [4]	Yes	Yes	Its nature is binary (extendable)
2	TSL [3]	Starvation	Yes	In case of processor support of the TSL instruction
3	SWAP [2]	Starvation	Yes	In case of processor support of the SWAP instruction
4	Binary semaphore [4]	Yes	No	Accomplishment of two action SIGNAL and WAIT should be done in atomic and indissoluble (regarding mutual exclusion)
5	Monitors [3]	Yes	No	In case of programming language support of monitors

In programming language solutions that is monitors, it should be noted that some of common programming languages do not support them (e.g., C and Pascal). Therefore, using this solution depends on the supporting conditions of programming language which is not always possible. Also, usage of them are limited to mono-processor or multi-processor with mutual memory. On the other hand, semaphores can control most of essential conditions in processes competitive conditions that is mutual exclusion, promotion and can control limit waiting and also can accomplished by indefinite number of processes and no busy waiting or live lock occur during their usage. In addition, semaphores are can be accomplished in all of systems or on the other word their covering area are wider than other solutions. It should be noted that semaphores need the special ability of programmer in synchronizing to control their complexity. In this case choosing the initial amount for semaphores is difficult.

According to Table 2, two actions of WAIT and SIGNAL in semaphores should be accomplished in form of atomic and indissoluble. This is because of the need for mutual exclusion in this action and each moment just one process can have the permission to manipulate the semaphores using one of these two actions.

4. Conclusions

This paper presents a survey of current algorithms for management of the problem of concurrency in operating systems. The existing methods are classified into four categories: 1) software-based algorithms, 2) hardware-based algorithms, 3) based operating system, and 4) based on message passing. Furthermore, it provides a comparison of algorithms capabilities. For this, two different studies are conducted to compare algorithm characteristics within their categories (Intra-group analysis) and between different categories (Inter-group analysis). Table 3 summarizes these results. In the second study comparison analysis, after comparing current algorithms capabilities it was concluded that semaphores considers all conditions of mutual exclusion, promotion and limit waiting and do not have busy waiting problem. On the other hand, they are capable to be accomplished on all systems because of their support by operating systems. The only problem that seems to be hidden in semaphores is mutual exclusion regarding in two actions WAIT and SIGNAL. Indissolubility of these two actions is able to be managed using three methods presented in [3], and [10-12], which are summarized as bellows:

1) **Software method:** In this method, software solutions such as Peterson can be used. However, this solution has busy waiting problem.
2) **Hardware solution of pause activation and deactivation:** because semaphores are supported by operating system so all pauses can be deactivated while testing the amount of semaphores, updating, sleeping and waking up the processes, but this method is useful when the system is mono-processor.
3) **Using the TSL instruction:** if the system is multi-processing, this solution can be used. It should be noted that the processor may not support this instruction and it can be accomplished using a high-level language.

Table 3 shows the classification of algorithms and represents the advantages and disadvantages of atomic SIGNAL and WAIT accomplishment.

Table 3. Advantages and Disadvantages of Different Categories

No.	Method	Advantage	Disadvantage
1	Software solution	Regarding mutual exclusion	Low speed and reducing competency
2	Hardware solution of pause activation and deactivation	Regarding mutual exclusion and High speed	Accomplishment just on mono-processor systems
3	Hardware solution of using TSL instruction	Regarding mutual exclusion and High speed	Possibility of not supporting the instruction by processor

According to Table 3, semaphores are suggested in the following circumstances:

1) If the system is mono-processor, using the hardware methods of pause activation and deactivation which consider the mutual exclusion and also have reasonable speed.
2) If the system is multi-processor and processor supports the TSL instruction, hardware solution based on the TSL instruction which consider the mutual exclusion but have the busy waiting can be used.
3) If the system is multi-processing and does not support the TSL instruction, using the software based solution can be considered which manage the mutual exclusion but has busy waiting problem.

Using semaphores in case of indissolubility of two actions WAIT and SIGNAL, just in one case may have busy

waiting (if the system be multi-processing and does not support TSL instruction) . In these two forms do not have busy waiting and have proper speed (if the system is multi-processing and processor support the TSL instruction and or the system is mono-processing). Therefore, the comprehensive use of semaphores just in certain condition that the system does not support the TSL instruction and is multi-processing has busy waiting and are able to be accomplished on all systems, this can occur if:

- Software and hardware based solutions in the second study (the Peterson, TSL, and SWAP methods) have always busy waiting problem.
- Programming language based solutions are not supported in some programming languages (e.g. C, Pascal) and just can be used in mono-processor or multi-processor systems with mutual memory.

According to the comparison analysis results, it was realized that semaphores are more competent than other solutions for concurrency and synchronization control in operating systems. However except in case of copping with busy waiting, in most of occasions have acceptable performance in operating systems.

References

1. W.Stallings, Operating System Internal And Design Principles, Prentice Hall, 2009 7th Edition, March 2011.
2. A.Silberchatz, B.Galvin, G.Gange, Operating System Concepts, Wiley, 9th Edition, December 2014.
3. A.s Tanenbaum A.s Woodhull, Operating Systems Design & Implementation, Pearson Prentice Hall, 3rd Edition, January 2009.
4. A.s Tanenbaum, Modern Operating Systems, Pearson Education, 3rd Edition, 2008.
5. Clay Breshears, The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications, Oreilly, 2009.
6. Leslie Lamport, A New Solution Of Dijkstra's Concurrent Programming Problem, Massachusetts Computer Associates , Communication of the ACM, Vol. 17, No. 8, 1974.
7. M. Ben-Ari, Principles of Concurrent and Distributed Programming, Addison-Wesley, 2th Edition , February 2006.
8. Sibsankar Haldar, Alex Aravind, Operating Systems, Pearson, November 2010.
9. Peter W. O'Hearn, Concurrency and Local Reasoning,In Proc. 13th European Symposium on Programming, Spain, ESOP 2004.
10. Andrew D. Birrell, Implementing Condition Variables with Semaphores, Microsoft Research, Silicon Valley, January 2003.
11. Matt Welsh, Systems Programming and Machine Organization, Harvard University, November 2009.
12. CarstenGriwodz, Semaphores and other Wait-and-Signal mechanisms, University of Oslo