DIFFERENT ALGORITHM OF CRITICAL SECTION

What do u mean by Critical Section?

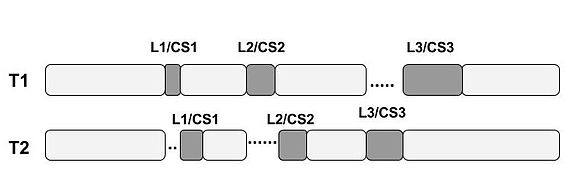
When Shared resources are accessed they are need to be protected in a such a way that they avoid the concurrent access.

This Protected section is the Critical Section.

How to Implement the critical section?

The implementation of critical section vary among different operating system.

So critical section will usually terminate in finite time, thread, task, or process will have to wait for a fixed time to enter it (bounded waiting). To ensure exclusive use of critical sections some synchronization mechanism is required at the entry and exit of the program.



Uses of Critical Sections

Kernel- level Critical sections

Critical sections in data structures

Critical sections in computer networking

What is Critical section problem in operating system?

Critical Section is the part of a program which tries to access shared resource. It can’t be executed by more than one process at the same time.

The Critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

In Order to synchronize the cooperative processes, our main task is to solve the critical section problem. Solution should satisfied following consitions:

Primary

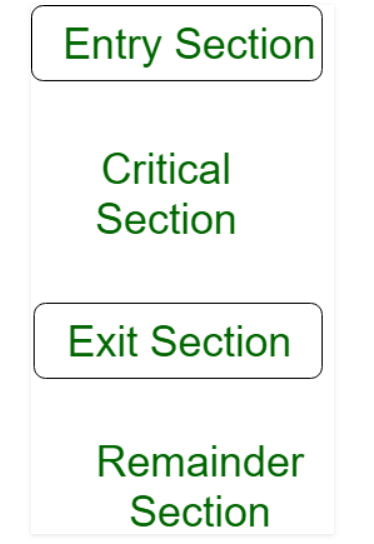
Mutual Exclusion (if one process is executing inside critical section then the other process must not enter in the critical section)

Progress

Secondary

Bounded Waiting

Architectural Neutrality ( if one Solution is working well on one architecture then it should also run on the other ones as well.)



Types of Algorithm For Critical Section Problem in Operating System

Peterson’s Algorithm For Critical Section Problem-

. Software is based on solution to Critical Section Problem

. Doesn’t work on modern architectures.

.It’s for 2 processes which alternate execution between then critical section and remainder section. Say P1 is the first Process and P2 is the second process.

. The 2 processes should share 2 data items with each other.

int Turn

Boolean flag[2]

Turn – It indicates the process who should enter into its critical section.

Flag Array – It tells whether a process is ready to enter its critical section. Let flag[0] indicate process P1. If flag[0] = true , then Process P1 is ready to execute in its critical section. flag[1] indicates process P2. If flag[1] = true, then Process P2 is ready to execute in its critical section.

EXAMPLE:-

Do {

Flag[i]=true;

Turn = j;

While (flag[j]&&turn ==j);

Critical Section

Flag[i]=false;

Remainder Section

} while(true);

. First , p1 sets flag[0] true, then sets turn to j. So that if P2 wants to enter Critical Section, it can do so.

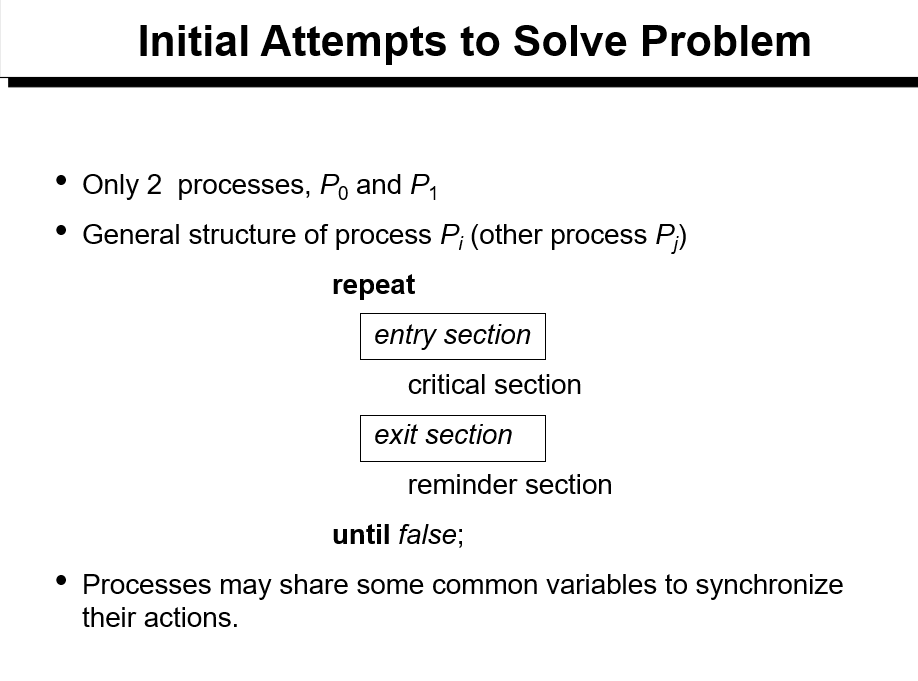
. If p1,p2 try to execute at same time , then turn is first changed to I, then j or it could be vice-versa. But, the important point is , only one of these 2 process is allowed to enter its critical section. The second value gets overwritten.

Features

. Does not require any special hardware.

. Uses Busy waiting (Spinlock).

What is Race Around Condition?

If many kernel processes in OS, It may lead to race around condition.

ALGORITHM 1

. Shared Variables:

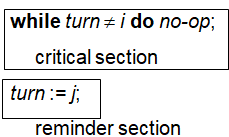
Var turn:(0..1);

Initially turn = 0

turn-I -> pi can enter its Critical Section

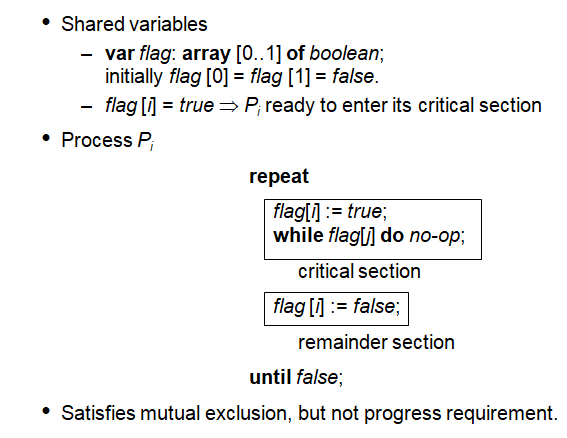
. Process Pi

Repeat

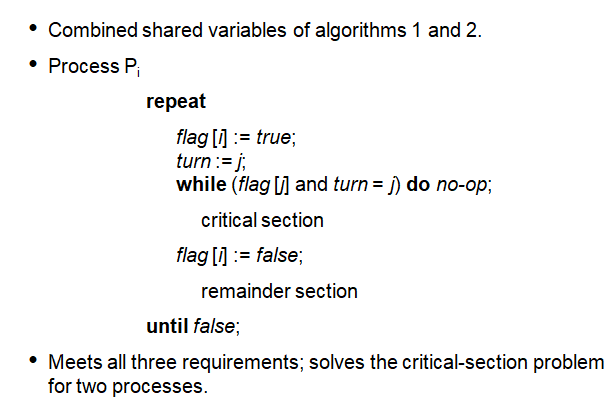


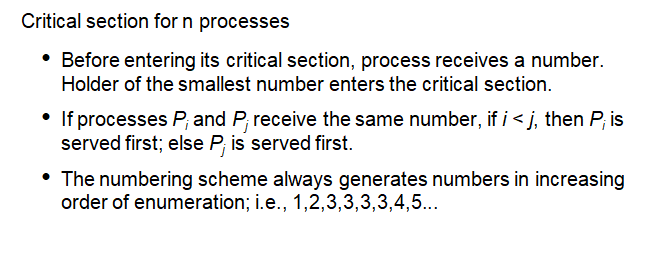
. Satisfies mutual exclusion , but not progress

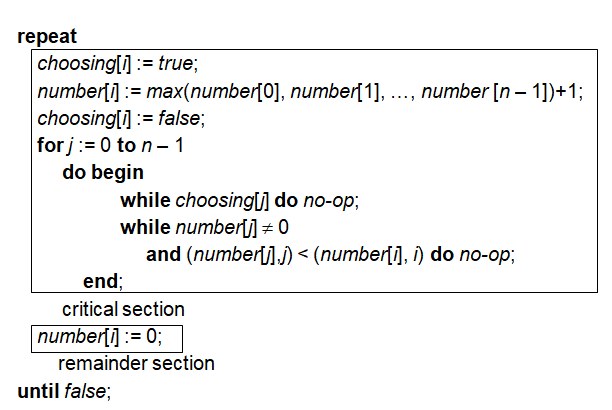
ALOGORITHM-2



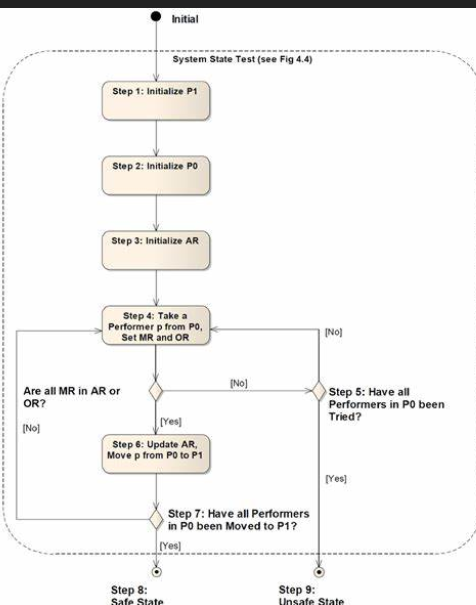
ALOGORITHM-3



BAKERY ALOGORITHM



BANKER’S ALGORITHM



The Banker algorithm is also known as detection algorithm.

It’s a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources , and then makes an “s-state” Check to test for possible deadlock conditions for all other pending activities.

It was developed for the operating system and originally described in EWD108.

When a new process enters a system, it must declare the maximum number of instances of each resource type that it may every claim; clearly, that number may not exceed the total number of resources in the system. It must return them in a finite amount time.

For the Banker’s Algorithm to work, three needs u need to know:

HOW much of each resource each process could possibly reuest[MAX]

HOW much of each resource each process is currently holding[Allocated]

How much of each resource the system currently has available[AVAILABLE]

Resources may be allocated to a process only if the amount of resources requested is less than or equal to the amount available; otherwise, the process waits until resources are available.

Some of the resources that are tracked in real systems are [memory](https://en.wikipedia.org/wiki/Memory_(computers)), [semaphores](https://en.wikipedia.org/wiki/Semaphore_(programming)) and [interface](https://en.wikipedia.org/wiki/Interface_(computer_science)) access.

Basic data structures to be maintained to implement the Banker's Algorithm:

Let {\displaystyle n} be the number of processes in the system and {\displaystyle m} be the number of resource types. Then we need the following data structures:

Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type Rj available.

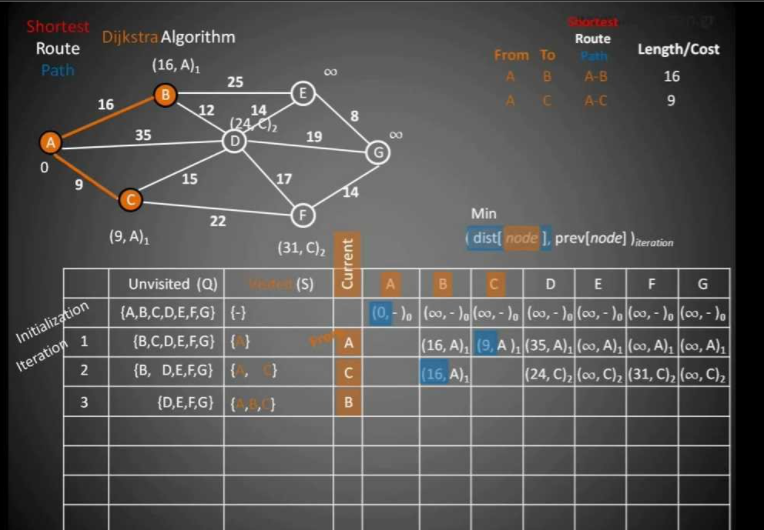
Max: An {\displaystyle n} × {\displaystyle m} matrix defines the maximum demand of each process. If Max[i,j] = k, then Pi may request at most k instances of resource type Rj.

Allocation: An {\displaystyle n} × {\displaystyle m} matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process Pi is currently allocated k instances of resource type Rj.

Need: An {\displaystyle n} × {\displaystyle m} matrix indicates the remaining resource need of each process. If Need[i,j] = k, then Pi may need k more instances of resource type Rj to complete the task.

Note: Need[i,j] = Max[i,j] - Allocation[i,j]. n=m-a.

DEADLOCK DETECTION ALGORITHM(Shoshani and Coffman



Suppose if system doesn’t have deadlock prevention and no deadlock scheme, then it needs a deadlock detection scheme with recovery from deadlock capability. An Algorithm is needed which will determine whether the system has entered a dead lock state. This algorithm must be invoked periodically

The algorithm is as follows:

Initialize Work = Available

For i=1 to n do

If Allocation(i) = 0 then Finish[i]= true else Finish[i] =false

2.Search an I such that

Finish[i] = false and request(i)<= Work

If no such I can be found , go to step 4.

3.For that I found in step2 do:

Work = Work + Allocation(i)

Finish[i] = true

Go to step 2.

If Finish[i] /= true for some I then the system is in deadlock state else the system is safe

ROUND ROBIN SCHEDULING ALGORITHM

It’s a preemptive scheduling algorithm and there is fixed time assign to all processes and time is called time quantum.

All the processes can execute only until their time quantum and then leave the CPU and give a chance to other processes to complete and give a chance to other processes to completer their execution according to time quantum.

FORMULA

Turn Around Time = Completion Time – Arrival Time

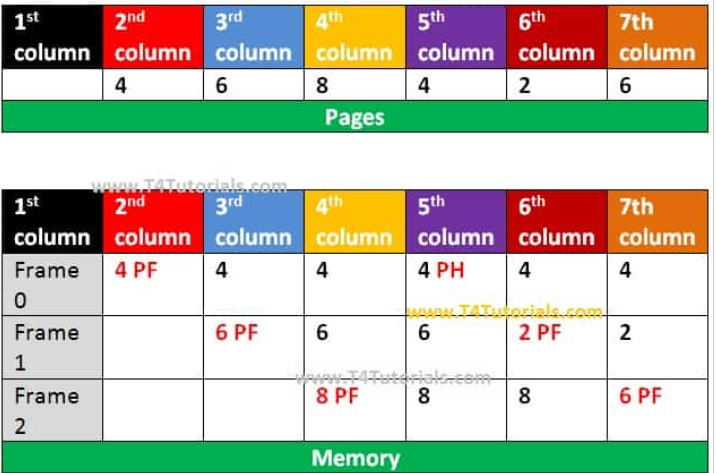
Waiting Time(W.T.) : Time Difference between the turnaround and the burst time.

Waiting Time = Turn Around Time – Burst Time

|  |  |
| --- | --- |
| Advantage | Disadvantages |
| .It avoid starvation or convoy effect. | . It utilize more time on context switching which is not good in some cases. |
| .All the jobs get a fair and efficient allocation in CPU | . Output of processor Iin this algo. Will be reduced , if slicing time of OS is low |
| .It don’t care about priority of the processes. | . if time quantum is low , IT will increases the context switching time |
| . It gives the best performance in terms of average response time. | . It can lead to dec. the comprehension. |
| . It is easily implementable on the system because it does not depend upon burst time | . We can’t set priorities for the processes. |
| . Promotes Context switching to save the states of the preempted processes. | .IT heavily depend on time quantum of the processes. |
| . Worst case response time for the process , if we know the total number of processes on the running queue. | . It’s a difficult to decide a correct time quantum to increase the efficiency  and speed. |

LEAST RECENTLY USED PAGE REPLACEMENT ALGORITHM IN OPERATING SYSTEM

It works on a prediction that the pages have been used more times in the last few instruction will probably be used again and again



1 column: First of all, all memory is free.

2 column: Page 4 is loaded on memory. The page fault occurs because page 4 is already not on memory.

3 column: Page 6 is loaded on memory. The page fault occurs.

4 column: Page 8 is loaded on memory. The page fault occurs.

5 column: Page 4 is already loaded on memory. Page Hit occurs.

6 column: Page 2 is loaded on memory. The page fault occurs.

7 column: Page 6 is loaded on memory. The page fault occurs.

SYNCHRONIZATION

Its task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.

Specially needed in a multi-process system when multiple processes are running together.

IT can also lead to the inconsistency of shad data . So the change made by one process not necessarily reflected when other processes accessed the same shared data. To avoid this type of inconsistency of data the processes nedd to be synchronized with each other.



Dekker’s Algorithm

The first Correct solution problem:

It allows two threads to share a single-use resource without conflict, using only shared memory for communication.

It avoids the strict alternation of a turn-taking algorithm, and was one of the first mutual exclusion algorithms to be invented.

There are many version of this algorithm nut the current one is (5th version) the one that satisfies

Mutual Exclusion

Progress

Bounded waiting (Conditions and the most efficient one)

Dekker’s Solution, ensures mutual exclusion between two processes only , It could be extended to more than two processes with the proper use of arrays and variables.

Algorithm : It requires a Boolean values , integer varaiable

Example:-

Var flag: array[0..1] of Boolean;

turn: 0..1;

repeat

flag[i] := true;

while flag[j] do

if turn = j then

begin

flag[i] : = false;

while turn = j do no-op;

flag[i] ; = true;

end;

critical section

turn ;=j;

falg[i] := false;

remainder section

until false;

PRIORITY SCHEDULING ALGORITHM:

It is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority depends upon memory requirements, time requirements, etc.

There are two types of Priority Scheduling

. Preemptive Scheduling

. Non-Preemptive Scheduling

Advantages

. Easy to use

. Does not need to wait

. Provides a good mechanism

. Suitable for applications with fluctuating tie and resource needed.

Disadvantages

. During system crashes low priority processes get lost

. Lower priority processes may starve or be postponed for an indefinite time

. processes will be blocked if its ready to run but has to wait for the CPU.

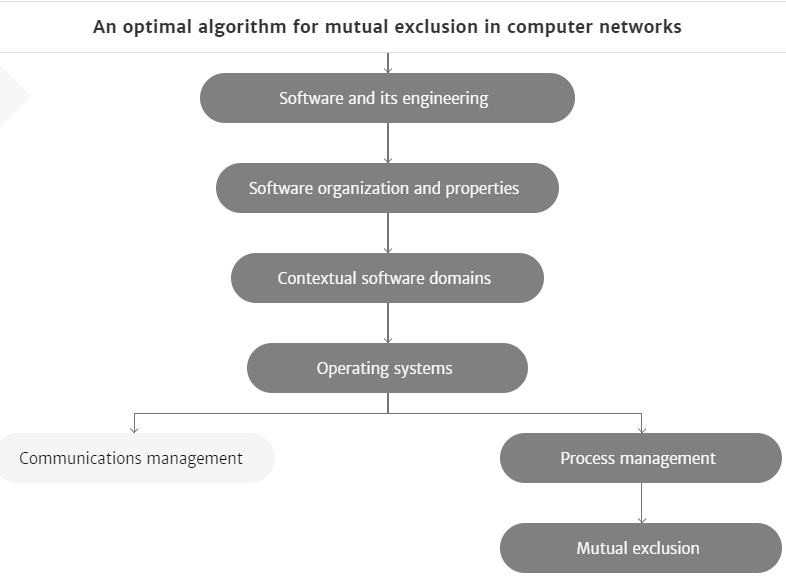
Abstract

A new algo. is proposed to allow K simultaneous entries into a critical section in distributed . The proposed algorithm needs in the worst case half as many message exchanges as in a recently published algorithm (k.Raymond, Inform, Process. Lett. 30(1989)189-193) . A simplistic average case analysis of the proposed algorithm is presented which shows that we get almost 50% savings in number of messages in the average case also.

Keywords

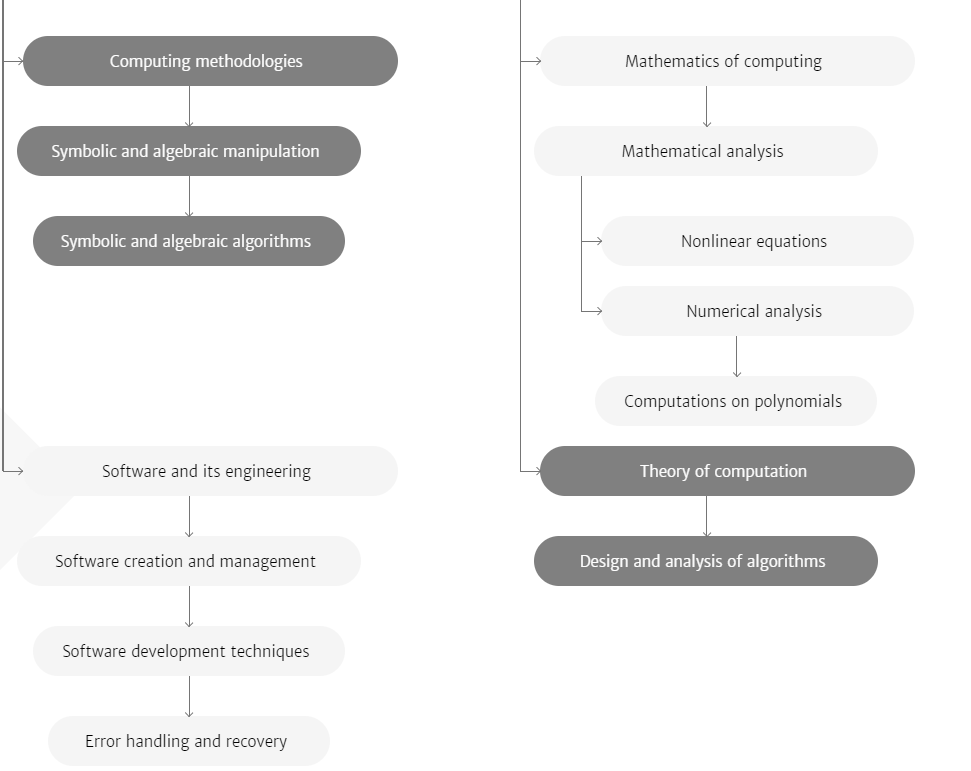
Distributed system; mutual exclusion

An Optimal algorithm for mutual exclusion



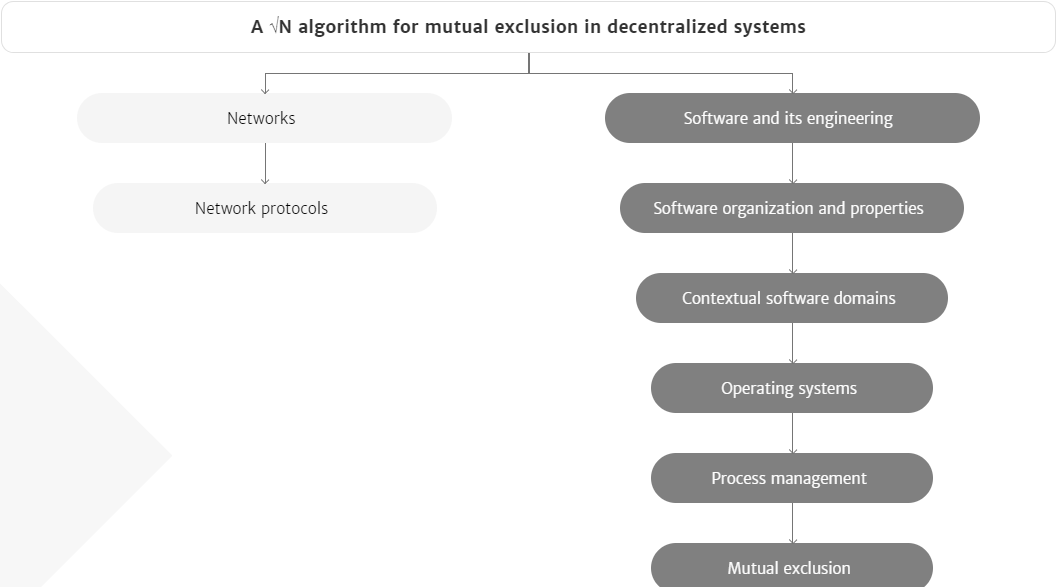
An algorithm is proposed that created mutual exclusion in a computer network whose nodes communicate only by message and do not share memory. The algorithm sends only 2\*(N-1) messages, where N is the number of nodes in the network per critical section invocation. This number of messages is at a minimum if parallel, distributed, symmetric control is used; hence, the algorithm is optimal in this respect. The time needed to achieve mutual exclusion is also minimal under some general assumptions

A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)



A √N algorithm for mutual exclusion in decentralized systems

An algorithm is presented that uses only c√N messages to create mutual exclusion in a computer network , where N is the number of nodes and c a constant between 3 and 5. The algorithm is symmetric and allows fully parallel operation.



New analytical model to find the max. tooth root stess and critical section location of spu gear;

New analytical model is based on mechanics theory.

It can determine the critical section location accurately and quickly by solving the extreme value.

Moment stress and compressive axial stress are taken into account in the model with the accurate profile equation.

Its more reliable than international standards.

. Accurate calculation of the maximum tooth root stress (TRS) and Critical location (CSL) provides a basis for predicting and improving gear performance.

. In current research, finite element methods (FEM) model and experimental test methods (ET) model can obtain accurate results but need large computational resources and time.

. Results from ISO 6336.2006 (ISO) model and AGMA 2101-D04(AGMA) model are obtained conveniently but sometimes not reliable.

. Results of the spur gear in five cases with different parameters are obtained and compared to those of the FEM, ISO and AGMA models.

KEYWORDS

Maximum tooth root stress; Critical section location; Analytical method; Involute spur gear

Asynchronous Message-Passing Distributed Algorithm(local critical section problem)

It’s a generalized local version of critical problem including mutual exclusion, mutual inclusion, k-mutual exclusion and I-mutual inclusion.

Mutual exclusion problem is a fundamental process synchronization problem in concurrent systems [1,2,3].

Proposed Algorithm for the Generalized local li-Mutual inclusion