CHAPTER 5
COLORED CRITICAL SECTIONS AND FAULT TOLERANCE

5.1. INTRODUCTION

The various services offered by a server modelled as the concurrent monitor can potentially be spread across various physical processor nodes. Moreover, implementation of load balancing techniques would lead to constant migration of various services within the entire system. We need to devise a mechanism by which the client requests the required service without too much dependence on the book-keeping abilities of the node on which the main server block is placed i.e., primary server node.

A typical service call specifies the server name and the service name. The service name in this case is the name of the accept statement, and the set of parameters. Providing such a capability alone would lead to the serialization of all requests at the primary server node. This node has to locate the node on which the desired service is present and re-direct the service call to that node. This is a centralized solution which leads to a bottleneck at the primary server node.

The author presents an alternative solution to solve this bottleneck (Gopal 90c). Each of the blocks of the server that can be distributed to other nodes in the system has a unique color. The service call specifies the color of the distributable segment containing the service it requires in addition to the usual information. The operating system provides the mapping between the color and the physical address of the corresponding node. Moreover, the solution is totally message based thus providing a flexible implementation on workstation model providing location transparency to the client. This also facilitates the implementation of a variety of load balancing techniques discussed in the next chapter.
Token passing algorithms have gained immense popularity due to their simplicity and ease of maintenance. However, the functioning of the entire system is dependent on the well being of an entity called 'token'. The system ceases to function if the token is lost. There are algorithms suggested for improving the fault tolerant capabilities of the system with respect to the token. A new algorithm to achieve fault tolerance is proposed. The algorithm is implemented on a simulated network of nodes based on the IEEE 802.5 standard (Norman 1983), (Tanenbaum 1989).

5.2. COLORED CRITICAL SECTIONS

The algorithm is an extension of Ricart and Agrawala's algorithm requiring 0 to N messages where N is the number of processes in the system (Gopal 90c). The messages used are of two distinct types namely 'request' and 'access'.

The data structures used in the algorithm are:

**TYPE**

color = (clr1, clr2, clr3,..., clrN); processes = 1..n;

**VAR**

tokens : array [color] of array [processes] of integer;
requests : array [color] of array [processes] of integer;
clock : integer; { initialised to zero }
token_present : array [color] of boolean;
token_held : array [color] of boolean;
reccclr : color;
The boolean array token_present may be initialised to false in all processes except the initial process holding the tokens. It is to be noted that all the tokens need not be in the same process at the time of initialisation of the system. In such cases this array may be appropriately initialised. The arrays, tokens and requests, are both initialised to zero.

This solution increases the potential concurrency but introduces asymmetry in the actions performed by each constituent process. This is because of the fact that, each process knows the color of the token that leads to the service required. As an illustration, the actions performed by a process 'i' that requires either a token clr3 or clr7 performs the following set of actions before entering the critical section. The function 'wait' waits for a signal named SIGHOK from the THREAD 3 of the algorithm. A parameter (recdclr) is also passed with the signal. Signal received by THREAD 1 when it is not waiting for one is ignored.

The proposed algorithm has three threads. These threads can execute concurrently to further enhance the performance subject to the property that lines of code enclosed between 'STARTATOMIC' and 'ENDATOMIC' in the thread must be excluded with the execution of such parts in other threads in the same process. This can be realised by the use of traditional solutions for mutual exclusion in shared memory systems such as semaphores.

As an illustration, the actions performed by a process 'i' that requires either a token clr3 or clr7 performs the following set of actions before entering the critical section. The function 'wait' waits for a signal named SIGHOK from the THREAD 3 of the algorithm. A parameter (recdclr) is also passed with the signal. Signal received by THREAD 1 when it is not waiting for one is ignored.
THREAD 1

<< GETTOKEN >>

if not (token_present[clr3]) and not(token_present[clr7])
then
begin
    clock := clock + 1;
    parbegin { concurrent broadcast }
    broadcast(request,clock,clr3,i);
    broadcast(request,clock,clr7,i);
    parend
    wait(SIGHOK,THREAD3,recdclr);
    token_held[recdclr] := true;
end;
else
STARTATOMIC
if token_present[clr3] then
    recdclr := clr3;
    token_held[recdclr] := true;
endif;
if token_present[clr7] then
    recdclr := clr7;
    token_held[recdclr] := true;
endif;
ENDATOMIC
endif;
if not(token_held[clr3]) and not(token_held[clr7]) then

goto GETTOKEN
endif;

<CRITICAL SECTION>

tokens[recdclr][i] := clock;

token_held[recdclr] := false;

for clr from clr1 to clrN do

for j from i+1 to n, 1 to j-1 do

if requests[clr][j] > tokens[clr][j] and token_present[clr] then

begin


token_present[clr] := false;

send(access,tokens,clr) to j;

end;

endif;

endfor;

for clr from clr1 to clrN do

for j from i+1 to n, 1 to j-1 do

if requests[clr][j] > tokens[clr][j] and token_present[clr]

then

begin


token_present[clr] := false;

send(access,tokens,clr) to j;

end;

endif;

endfor;

THREAD 2

when received(request,k,clr,j) do

requests[clr][j] := max(request[clr][j],k);

STARTATOMIC

if token_present[clr] and not(token_held[clr]) then

then


tokens[clr][i] := clock;

token_held[clr] := false;

endif;
ENDATOMIC

for clr from clrl to clrN do
    for j from i+1 to n, 1 to j-1 do
        if request[clr][j] > tokens[clr][j] and token_present[clr] then
            begin
                token_present[clr] := false;
                send(access, tokens, clr) to j;
            end;
        endif;
    endif;
endif;

THREAD 3

when received(access, tokens, clr) do
    token_present[clr] := true;
    if token_present[clr3] or token_present[clr7] then
        signal(SIGHOK, THREAD1, clr);
    endif;

5.3. CORRECTNESS OF THE ALGORITHM

The following assumptions are made about the communication network connecting the processes

- The network is fully linked
- The transmission is error free
- The delay is variable
- The order of receiving the messages may be different from that in which they were transmitted

To establish the correctness of the algorithm the following properties have to be established.

- Mutual Exclusion
- Fairness
- Absence of Deadlock

As there is only one token for a particular color in the system and atmost one process would have the token present value corresponding color set, the mutual exclusion property of the algorithm is established.

As the transmission delays in the system are finite, the requests are duly recorded in a finite time. The 'THREAD 1' would then service the request by appropriate turn in a finite time. Thus every process gains the required token in a finite time thus ensuring fairness of the algorithm.

Let us assume that all the processes wish to enter a particular critical section but are all halted as none of them possesses the corresponding token. Obviously the token is in transit. From the assumptions made on the underlying network it follows that after a finite time the token reaches one of the processes and unblocks it. Thus there cannot be a deadlock in the system.

Eventhough the above discussion presupposes several aspects of the connecting network, in reality several faults do occur in the network. The nature of the faults is
potentially huge and they can occur in several combinations. The nature of the faults relevant to this work are outlined below.

5.4. NATURE OF FAULTS

A variety of faults can potentially occur in a token based network. The algorithms developed in this study take care of the following situations.

* A station in the network may breakdown holding the token along with it.

* A token may be lost in transit.

* A branded 'lost token' by the fault tolerant algorithm may reappear leading to duplicate tokens on the network.

5.4.1 Existing Fault Tolerant Algorithms

The following techniques are popularly employed in token passing networks for achieving fault tolerance (Raynal 1988)

a) Misra’s Algorithm

b) Timer Algorithm

5.4.1.1 Misra’s Algorithm

This algorithm uses two tokens ‘ping’ and ‘pong’. Associated with them are two numbers ‘nbping’ and ‘nbppong’ respectively. These numbers are equal in value but opposite in sign.

Thus nbping + nbpong = 0

Initialies nbping to +1 and nbpong to -1.
Each process \( P_i \) has an integer \( M_i \) initialised to 0.
when received \((\text{ping}, \text{nbping})\) do if \( M_i = \text{nbping} \) then
begin
    regenerate pong
    \( \text{nbping} = \text{nbping} + 1; \)
    \( \text{nbpong} = -\text{nbping}; \)
    end
else
    \( M_i = \text{nbping}; \)
enddo
when received \((\text{pong},\text{nbpong})\) do
    \(<\text{as before, interchange ping and pong}>\)
enddo
when meeting \((\text{ping},\text{pong})\) do
    \( \text{nbping} = \text{nbping} + 1; \)
    \( \text{nbpong} = \text{nbpong} - 1; \)
enddo

The values of the counters can be limited by limiting the number of processes
to \( n \) and performing a modulo arithmetic. The major disadvantage could be the
requirement of a multi-threaded process.

5.4.1.2 Timer Algorithms

The algorithm time-outs to regenerate the token but the time estimate is a
function of the individual process number, thus making it dependent on the exact node
addresses.
5.4.2 The Tagged Token Algorithm

The study of fault tolerance in token ring networks led to the development and testing of a new algorithm on the simulated token ring network (Gopal 94b).

The key aspect of this algorithm is the estimation of the worst-case time. If there are ‘n’ nodes on the ring, then at node i, the time interval between two consecutive visits of the token in the worst-case would be the time taken for (n-1) stations to transmit a packet each to its immediate neighbour not in the direction of transmission + time for (n-1) absorptions and regenerations of the token. The worst-case time estimated by every station is an integral multiple of this time quantum. The worst-case time estimate is the same for all stations on the ring.

To prevent duplicate tokens, the token has a number attached to it. The token generating station broadcasts the token number to all the nodes on the network. If a node receives a token bearing a different number then it is deemed invalid.

Algorithm

* Note the time when the token passes node i and the token number.

* if the token number does not tally with the one expected then discard it.

* If the token’s next visit to node i is not within the worst-case time then regenerate the token presuming that the old token is lost.
* Broadcast the new token number to all stations on the network.

* On receipt of a broadcast message note the expected token number.

5.5. CONCLUSION

Colored critical sections provide location transparency of various service to the client process thus enabling the implementation of load balancing mechanisms. The new algorithm for fault tolerance is general, simple and easy to implement when compared to the existing algorithms. It was able to handle a variety of probabilities of fault occurrence. It is however, limited to the occurrence of the faults specified earlier.