**CS6378.003 – ADVANCED OPERATING SYSTEM**

**Project Report**

**Submitted By:**

**Team 2**

**(Akshay, Aniruddha, Darshan, Arnab)**

Our implementation of a mutual exclusion algorithm consists of steps from both Maekawa’s algorithm and Raymond’s algorithm. Both of these algorithm use messages to inform the other nodes/processes about the system status. Our aim was to design and implement a new algorithm, which proves faster and more efficient than Maekawa’s algorithm.

Maekawa’s algorithm for mutual exclusion uses 5 types of messages and Raymond’s algorithm uses types of messages. Simply combining these two algorithm will increase the number of messages manifold. Our main criteria was to perform mutual exclusion by implementing both quorum-based and token-based algorithm and also increase the efficiency of the algorithm, when compared to Maekawa’s.

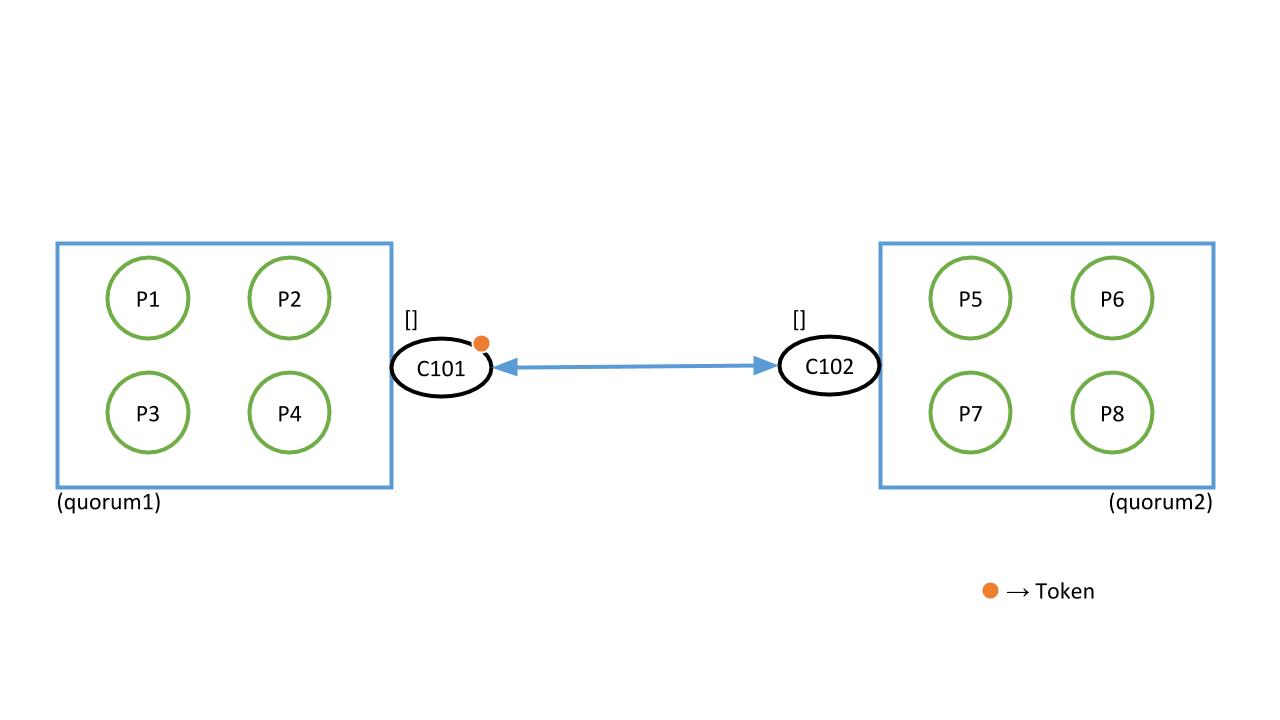
In our algorithm, we group nodes into quorums and then assign a controller node for a group of quorums. The purpose of controller node is to carry the token from one controller to other. The controller keeps a track of the internal processes i.e. total ordering of requests within the quorums. The controller nodes are not part of the quorums created.

**Working:**

* At the beginning, the participating nodes are grouped into quorums. Each group of quorums are assigned with a controller node. These controller nodes maintain a queue of their own, which keep a track of internal process requests.

Let us explain it by considering this example:

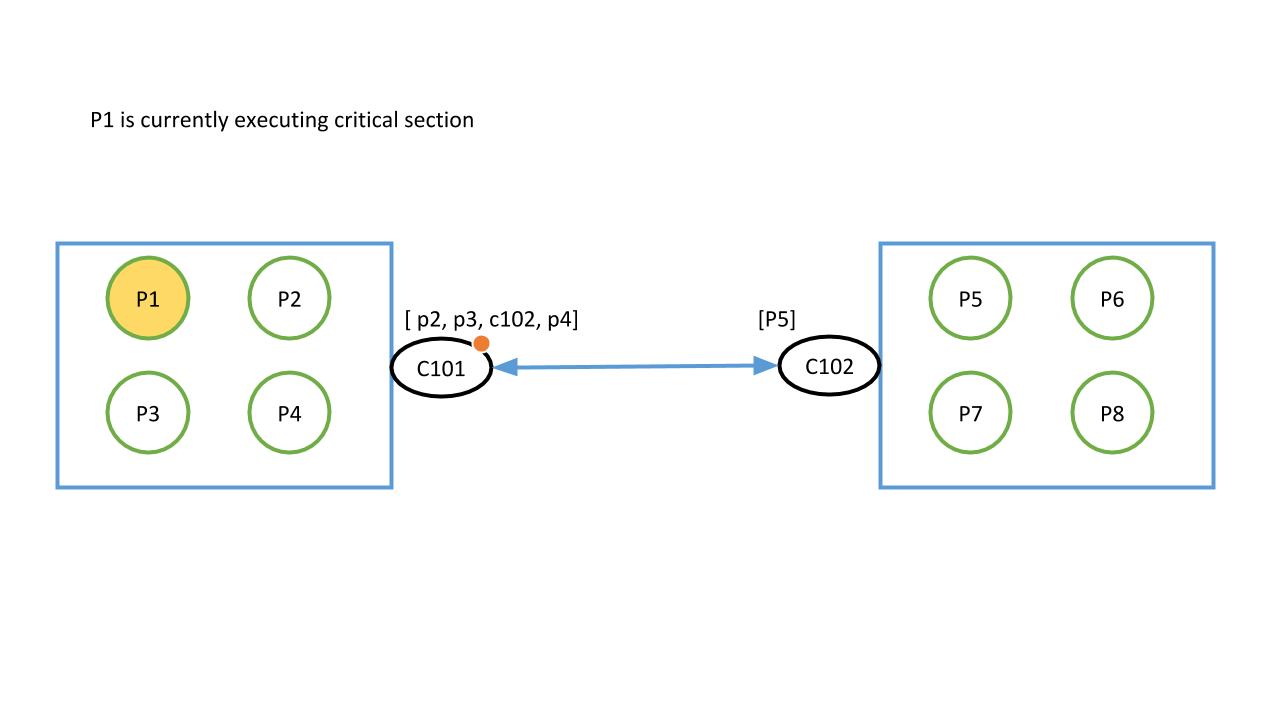
We have 8 processes, set into groups of 4 and each group has its own controller process. The layout will resemble like this:



As shown in the diagram, the token is currently with controller C101.

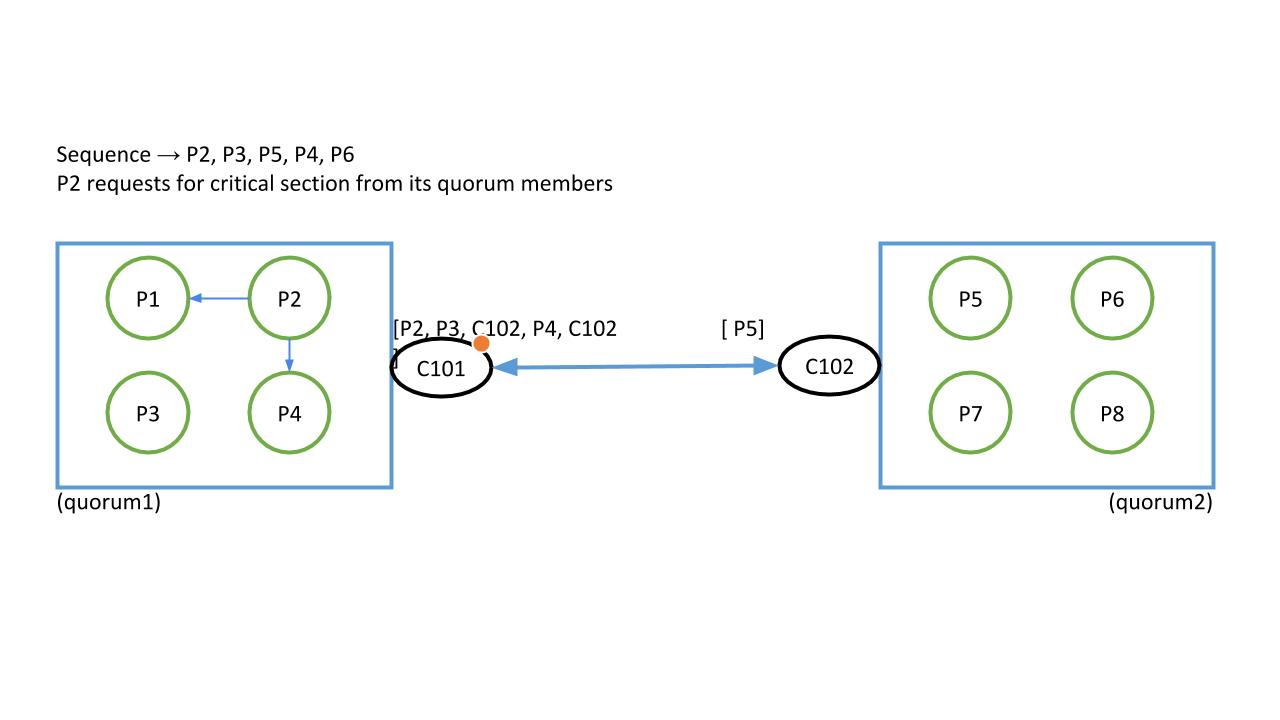
* Now, when all the processes start their execution, they need to enter the critical section one-by-one. In order to achieve that the process will ask for permission from its quorum members and also the controller node. The purpose of the controller node is to check whether it has the token or not. If not, it will insert the process number in its queue and ask other controllers for the token. A process can only enter its critical section if its controller node has the token.

For our example, let us consider that controller C1 and C2 each have 4 quorums Q1-Q4 and Q5-Q8 respectively. A process in Q1 wants to enter the critical section first. It will first ask permission from its quorum members. Once all the members have given permission and the controller has the possession of the token, it will enter the critical section.



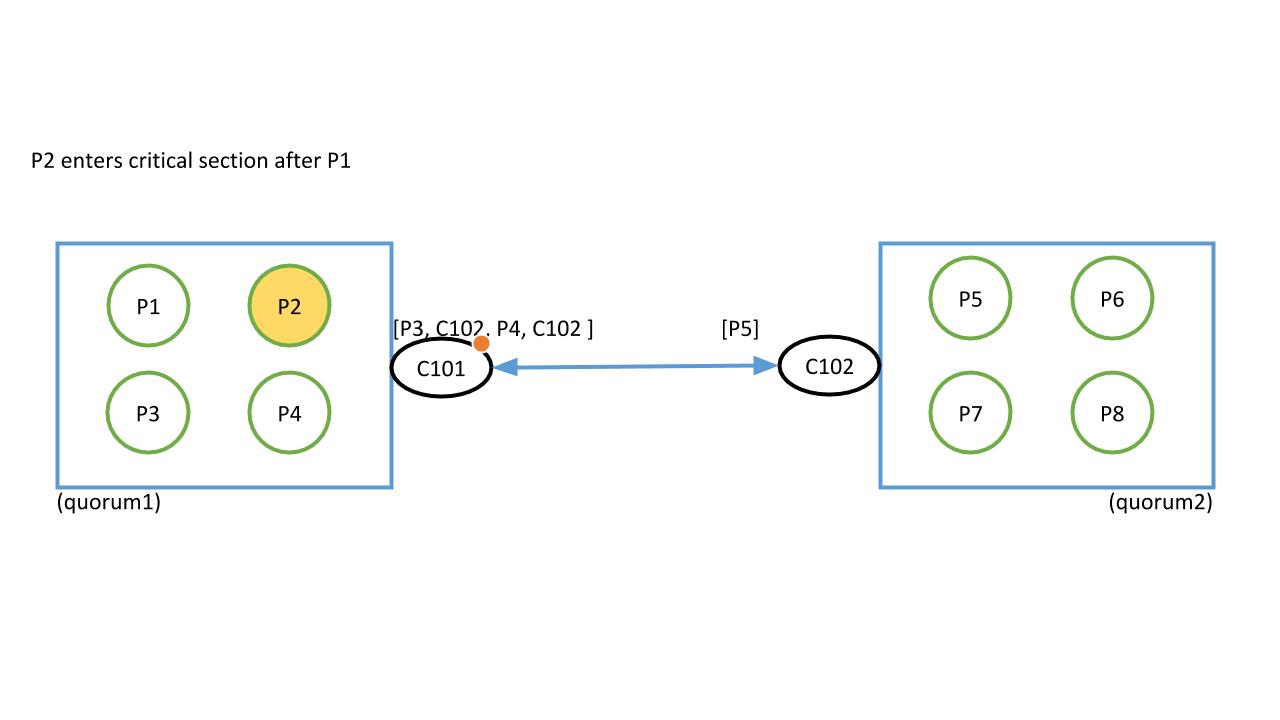
* Similarly, in the mean-time, if other processes ask for permission to enter critical section, it will enter the queue of the controller if any process is already in critical section. This also means that if process from other quorums want to enter the critical section, the controller nodes have to keep a track of the request and look for the token.

For example, suppose the following is the sequence in which the processes want to enter the critical section, when P1 is already in critical section. This is what it will look like:



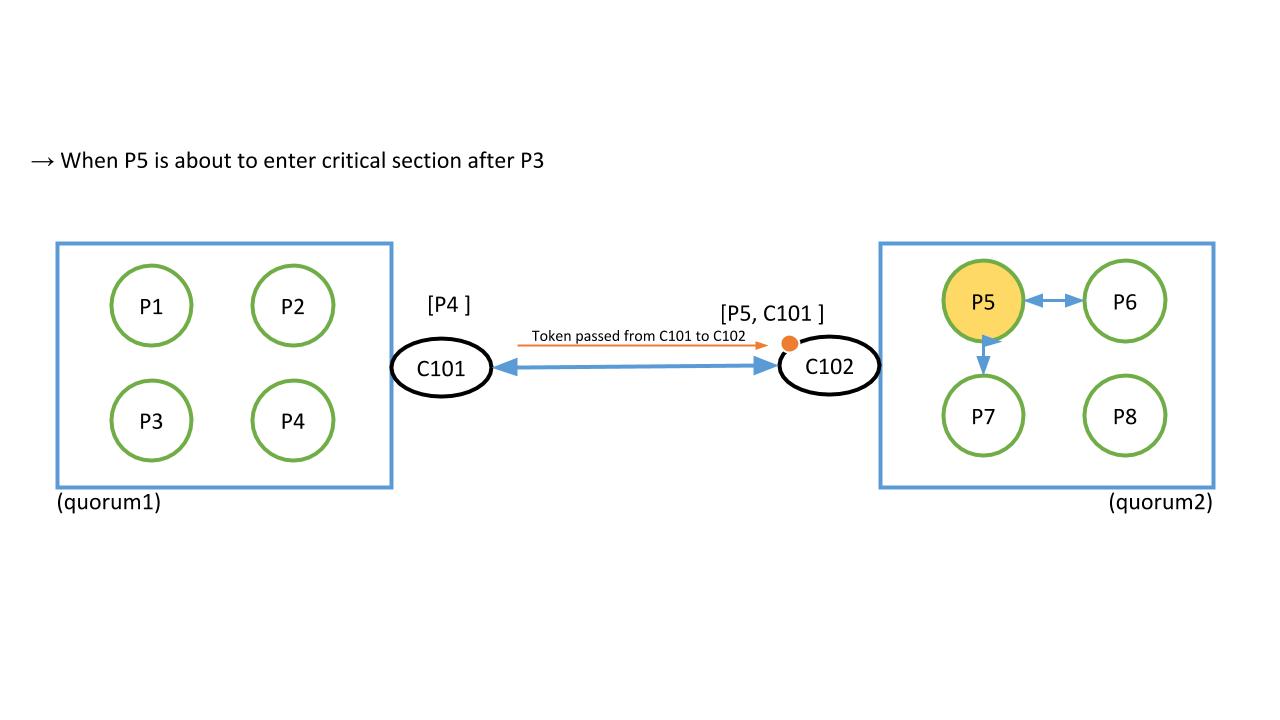
* Whenever one process finishes its execution, it will send release messages to all other processes in its quorum and also to controller node. Controller node checks which is the next process from its queue. If the process is in the same quorum, the token will still be with the same controller. If not, the token will be passed to the controller which has requested for it. When the new process is entering critical section, its request will be popped out of the queues of its own controller node and the same controller node request from other controllers as well.

We now look at 2 cases:



* This passing of token and execution of critical section by the processes will continue till all the requesting processes have finished the execution. That means that there are no more requests left in the controller processes’ queues.

This is the ideal case of execution of the algorithm.



Performance Metrics

Message Complexity –

Our algorithm requires lesser number of messages as compared to Maekawa’s algorithm when executed on the same number of nodes (processes).

Number of Messages required for Maekawa’s algorithm: 3(K-1)

Where K is size of quorum

Number of Messages required for our algorithm: 3(k’-1)+2(D-1)+2

Where,

K’ is size of quorum

D is number of controllers

Consider the example depicted earlier in the report.

Number of processes = 8

Number of controllers = 2

Maekawa’s algorithm –

K = 5

No. of messages = 3(5-1) = 12

Our algorithm –

K’ = 3

No. of messages = 3(3-1) +2(1) +2 = 10

Moreover, our implementation gives greater flexibility of design due to the controller concept.

**Conclusion:**

An algorithm for distributed mutual exclusion using a quorum and token based approach has been implemented. The performance of our algorithm is comparable to that of Maekawa’s quorum based algorithm. Our algorithm requires less number of messages as compared to Maekawa’s algorithm.