Safety Violation: The Safety Violation does not occur in Our Algorithm as we are deleting the request that we have processed in the critical section.

But it can be violated if a Process deletes a request which is not the one which was currently processed (Process deletes ‘any’ one request from its queue after executing critical section instead of the one having lowest timestamp)

Explanation:

Let’s take 2 processes P1 and P2

P1 and P2 both initially have following request messages in its queue : {1:P1 , 4:P2, 6:P1}

Step 1: P1 checks that the process with lowest timestamp belongs to itself. So it executes the critical section. Meanwhile P2 waits as it can see there exists a request which has a timestamp lower that what P2 has in the request queue.

Step2: Now P1 sends a release message to every one and delete any one request I:e 6:P1 from the queue

So P1 has this in its queue : {4:P2,1:P1}

P2 receives the release message from P1. Now lets say P2 removes any one request – ‘ 1:P1’ from the queue.

So P2 has this in its queue: { 6:P1,4:P2}

Now when P1 executes mutex it can see that it has a request in the queue which is lower than everyone so it goes into the critical section again.

Also, P2 will see that P2 has request 4:P2 in the queue which has the lowest timestamp so it also goes inside the critical section and since both P1 and P2 are inside the CS at the same time we have Safety Violation.

.\*Assuming that both P1 and P2 have received ack for their requests\*

Liveliness Problem : The liveliness problem occurs in the given Assignment version of Lamport Algorithm. The reason for that is we are deleting any one request of a particular Process from our queue when we receive a release message from a process.

Explanation:

Let’s take 2 processes P1 and P2

P1 and P2 both initially have following request messages in its queue : 1:P1 , 4:P2 and 6:P1

Step 1: P1 checks that the process with lowest timestamp belongs to itself. So it executes the critical section. Meanwhile P2 waits as it can see there exists a request which has a timestamp lower that what P2 has in the request queue.

Step2: Now P1 sends a release message to every one and delete its request 1:P1 from the queue.

So P1 has this in its queue : {4:P2,6:P1}

P2 receives the release message from P1. Now lets say P2 removes any one request – ‘ 6:P1’ from the queue.

So P2 has this in its queue: { 1:P1,4:P2}

Now when P1 executes mutex it will wait for P2 to send a release message as P2 is having the lowest timestamp according to its queue.

Also, P2 will again be waiting for P1 as in its queue P1 still has the lowest timestamp of ‘1’

In this way we have a deadlock in our system and liveliness is violated.