**Writer / Reader Protocol** Group: Meng Ju Wu, Yi Xu

When there is only writer involved in this system. According to the ME algorithm we use, Lamports Algorithm, whenever a server received a write request from client, it directly push the request into its local queue, update its local clock, and order the queue with timestamps. After that, it send request to other peer servers, then they store the request into their local queue as well and send back a ***WriteReply*** with updated timestamps. By doing the process above, all local queues will ultimately be identical and all servers are automatically synchronized. When a server found that the first request of its queue belongs to it, already got other’s replies, and ***ReaderInCriticalArea = 0*** (will be explained in next paragraph), it is able to access the critical data to update sum. During updating, critical area is locked result from that all of queues are having a same first request, but having receive a release message. After finishing update, the server send a ***WriteRelease*** message to other servers, and all of them pop the first request in their queue. The critical area is unlocked.

When there are writer and reader involve, the protocol of dealing with writer stay the same, but we introduce the other protocol for reader. Similarly, whenever a read request is gotten by a sever, it push it directly into queue, order queue, send request to other servers, and adjust its clock. Once a read request reach first of a queue, a server doesn’t have to wait for reply. Server directly send it to critical area and set ***ReaderInCriticalArea*** + 1. That is, if there are plenty of read request (n) following up, they all sent to critical area and ***ReaderInCriticalArea*** *= n.* Whenevera read request finished, server send a ***ReadRelease*** message to other servers, and they set ***ReaderInCriticalArea*** - 1. Once ***ReaderInCriticalArea =*** 0, that means instead of read request, the following request is a write request, and write request can access the critical area.

Using protocol described above, the system can avoid reading when a writing is on going, because whenever a write request reach critical area, the queue will be locked and allow no one to enter further. On the other hand, the system allow multiple reading concurrently, because the queue will not be locked when there’s reading inside. But a write request is not allowed because the ***ReaderInCriticalArea != 0.***