This program's processes achieve totally-ordered broadcasting using a similar approach to the Ricart-Agrawala algorithm. Processes broadcast a message to all other processes, which is treated as the request message in the Ricart-Agrawala algorithm. As a result, the processes determine the total order of messages by handling them the same way as a request to enter the critical section, corresponding to the server's need to make writes to the file system without interfering with other servers doing the same in the provided example. Upon receiving a request message, a process will determine if it has a higher priority based on the timestamp provided (and using the process id as a tiebreaker), sending a reply message if the requesting process does indeed have a higher priority. If the request has a lower priority, the message is buffered and deferred until the receiving process has gotten its turn in the critical section. The program therefore maintains mutual exclusion through total ordering of the messages by implementing the Ricart-Agrawala algorithm, which in this case ensures that mutually concurrent messages will be delivered in the same order for every process.

As highlighted in [1], "when two nodes are requesting concurrently, they do not know which of them made their request first because of the absence of timing information." As a result, "A tie-breaking scheme, representing a total ordering among requesting nodes, must be used." This is precisely the scenario when two processes attempt to broadcast messages concurrently in my program. By leveraging the Ricart-Agrawala algorithm, we address this challenge and ensure that even for concurrent message requests, there's a total ordering where all processes agree on the order of delivery.

Once a process finishes its broadcast of 100 messages, it no longer has any pending desire to enter the critical section, and it is done save for sending replies to any processes that still need access and subsequently send a request message. Additionally, if a process receives a request that has a causal dependency, the message can still be buffered until the causal predecessor arrives, whereupon the buffer is checked after receiving a message for any causal successors.

At the end of the program, the number of critical section executions are printed. Processes will defer all messages while they have a state of "HELD", and as a process will need the permission of all other processes to enter the critical section, we can ensure mutual exclusion. Processes will contest the request if they find their timestamp to be lower, or if their process id is lower in the case of a tie to ensure total ordering.

[1] Ellis, C. A., & Agarwal, S. (1984). The Ricart-Agrawala algorithm for mutual exclusion. ACM Transactions on Computer Systems (TOCS), 2(3), 245-265.