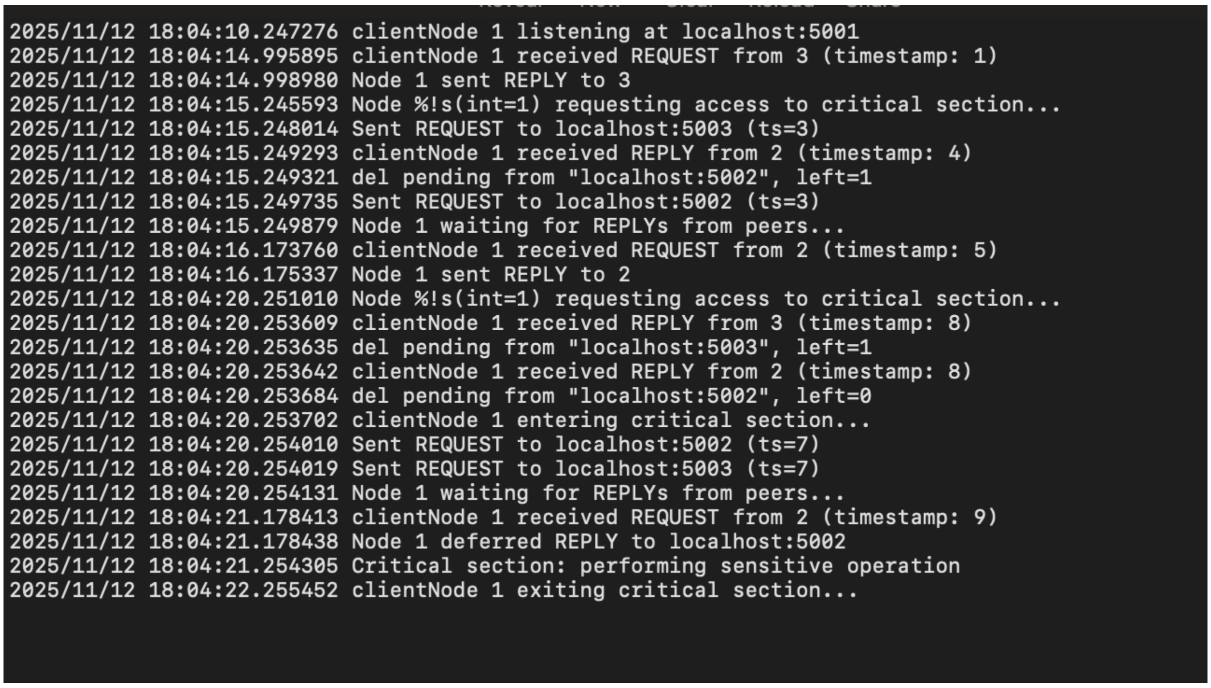
The system requirements are meet with the following specifications. R1 – System and Critical Section is emulated by a print statement. The nodes are instantiated emulating a distributed network where each node has access to send a request to the other nodes in the distributed system. This is ensured by using gRPC for network communication and passing a list of the peer nodes in the system.

Mutual Exclusion is ensured by a permission-based entry where each node only enters the CS if all other nodes has replied that I can access the critical section. The access reply can only happen if a node is not currently in the critical section, and Lamports timestamp is added to a request to ensure ordering and priority of the incoming request. Hence Ricart-Agrawala rules are enforced in RequestAccess; if a receiver is idle it replies immediately, if a receiver is in CS it defers the request to a deferred queue. If both are requesting the receiver compart the Lamport Timestamp, and the lower one wins. There is no possible overlap because everyone respects the same total order and each entrant must collect all permissions.

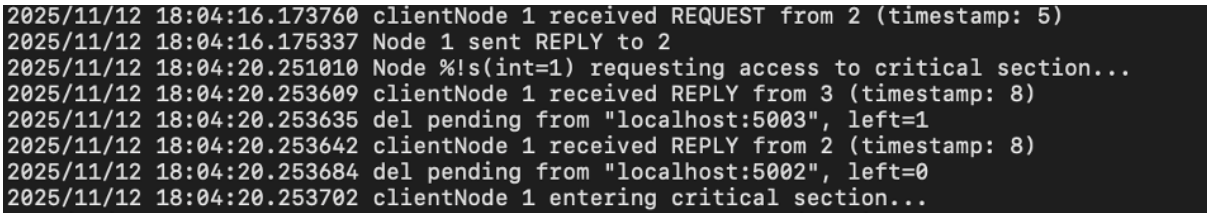
The system guarantees liveliness by ensuring that every node requesting access to the critical section will eventually be granted entry. Each request is assigned a Lamport timestamp, and if two requests have the same timestamp, the node ID is used as a tie-breaker. This creates a strict global order among all requests, ensuring that one node always has higher priority. When a node cannot immediately grant permission—because it is currently executing or waiting for the critical section—it adds the requester to a deferred queue. Once the node exits the critical section, it calls releaseDeferredRequests(), sending all pending replies and unblocking waiting nodes. Since the communication channels deliver messages in FIFO order, requests and replies are processed consistently, preventing circular waiting. Furthermore, each node only sends one request at a time, avoiding self-conflicts and ensuring progress. As a result, every node will eventually become the highest-priority requester and receive all the necessary replies to enter the critical section, thus satisfying the liveliness requirement.

The following screenshot shows an example of a sequence of messages that leads to a note entering the critical section.



```
2025/11/12 18:04:10.247276 clientNode 1 listening at localhost:5001
2025/11/12 18:04:14.995895 clientNode 1 received REQUEST from 3 (timestamp: 1)
2025/11/12 18:04:14.998980 Node 1 sent REPLY to 3
2025/11/12 18:04:15.245593 Node %!s(int=1) requesting access to critical section...
2025/11/12 18:04:15.248014 Sent REQUEST to localhost:5003 (ts=3)
2025/11/12 18:04:15.249293 clientNode 1 received REPLY from 2 (timestamp: 4)
2025/11/12 18:04:15.249321 del pending from "localhost:5002", left=1
2025/11/12 18:04:15.249735 Sent REQUEST to localhost:5002 (ts=3)
2025/11/12 18:04:15.249879 Node 1 waiting for REPLYs from peers...
2025/11/12 18:04:16.173760 clientNode 1 received REQUEST from 2 (timestamp: 5)
2025/11/12 18:04:16.175337 Node 1 sent REPLY to 2
2025/11/12 18:04:20.251010 Node %!s(int=1) requesting access to critical section...
2025/11/12 18:04:20.253609 clientNode 1 received REPLY from 3 (timestamp: 8)
2025/11/12 18:04:20.253635 del pending from "localhost:5003", left=1
2025/11/12 18:04:20.253642 clientNode 1 received REPLY from 2 (timestamp: 8)
2025/11/12 18:04:20.253684 del pending from "localhost:5002", left=0
2025/11/12 18:04:20.253702 clientNode 1 entering critical section...
2025/11/12 18:04:20.254010 Sent REQUEST to localhost:5002 (ts=7)
2025/11/12 18:04:20.254019 Sent REQUEST to localhost:5003 (ts=7)
2025/11/12 18:04:20.254131 Node 1 waiting for REPLYs from peers...
2025/11/12 18:04:21.178413 clientNode 1 received REQUEST from 2 (timestamp: 9)
2025/11/12 18:04:21.178438 Node 1 deferred REPLY to localhost:5002
2025/11/12 18:04:21.254305 Critical section: performing sensitive operation
2025/11/12 18:04:22.255452 clientNode 1 exiting critical section...
```

As seen a node sent a Request to It peers to enter the critical section. It then waits for all peers to reply. The waiting replies are gathered in a Map. When a node returns a reply that the node can enter the critical section, the given client is deleted from the map. This can be seen in the log statements below:



```
2025/11/12 18:04:16.173760 clientNode 1 received REQUEST from 2 (timestamp: 5)
2025/11/12 18:04:16.175337 Node 1 sent REPLY to 2
2025/11/12 18:04:20.251010 Node %!s(int=1) requesting access to critical section...
2025/11/12 18:04:20.253609 clientNode 1 received REPLY from 3 (timestamp: 8)
2025/11/12 18:04:20.253635 del pending from "localhost:5003", left=1
2025/11/12 18:04:20.253642 clientNode 1 received REPLY from 2 (timestamp: 8)
2025/11/12 18:04:20.253684 del pending from "localhost:5002", left=0
2025/11/12 18:04:20.253702 clientNode 1 entering critical section...
```

This clearely shows how a node is waiting for reply for all clients, and only when all client has received a reply will the node enter the critical section. This also clearly demonstrates that the program can run with three concurrent clients.