Distributed Systems - Mandatory 4

Alexander Llewelyn Yan Fatt Soo & Aske Thomas Neye

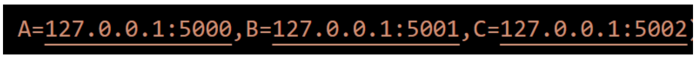
Github Repo:

https://github.com/askeneye/ricartagrawala

System Requirements

R1 (Spec): *Implement a system with a set of nodes and a Critical Section that represents a sensitive system operation. Any node may request access to the Critical Section at any time. In this exercise, the Critical Section can be emulated, for example, by a print statement or writing to a shared database on the network*

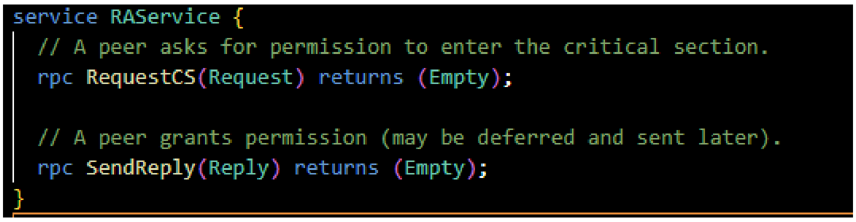
The system consists of three nodes **A**, **B** and **C**, hardcoded on its own port.



```
A=127.0.0.1:5000,B=127.0.0.1:5001,C=127.0.0.1:5002
```

Each node can request to enter the **Critical Section** — represented by a print/log message, which emulates a sensitive operation that must not overlap between nodes.

Each node is a gRPC server (for receiving requests and replies) and a gRPC client (to communicate with peers). Nodes exchange two RPCs:



```
service RAService {
  // A peer asks for permission to enter the critical section.
  rpc RequestCS(Request) returns (Empty);

  // A peer grants permission (may be deferred and sent later).
  rpc SendReply(Reply) returns (Empty);
}
```

R2 (Safety): *Only one node may enter the Critical Section at any time*

The system satisfies the **safety** requirement with a series of functions. In the **RequestCS()** function, a node defers any incoming request if it is currently executing or has already requested the Critical Section. These deferred requests are only granted after the node exits, as handled by **releaseDeferredReplies()**.
Additionally, a node can only enter the Critical Section with the **TryEnterCriticalSection()** function after receiving replies from *all* other peers, ensuring no overlap occurs.
This combination of deferred replies, Lamport timestamp ordering, and reply counting makes sure that only one node may be in the **Critical Section** at any time.

R3 (Liveliness): *Every node that requests access to the Critical Section will eventually gain access*

The **Liveliness** requirement is satisfied because the system makes sure that every request for the Critical Section is eventually granted. We handle this with the above-mentioned functions.
In the **RequestCS()** func, requests are never dropped, they are deferred if another node is currently in or waiting for the **Critical Section**. When a node exits, **releaseDeferredReplies()** sends all pending replies, allowing waiting nodes to proceed.
The use of Lamport timestamps, from the file lamportclock.go, makes sure we have an ordering of requests, preventing circular waiting or deadlocks. Each node maintains its own **Lamport clock** that increments with every event and message, which ensures global logical ordering of requests.

Discussion:

The system implements the **Ricart Agrawala mutual exclusion algorithm**, using gRPC for passing messages and Lamport timestamps for ordering of requests.
Each node maintains local state (**requesting, inCS, deferredReplies**) and coordinates access to the **Critical Section** by exchanging **RequestCS** and **SendReply** messages.
The execution logs from Nodes A, B, and C demonstrate that the system satisfies both the **Safety (R2)** and **Liveliness (R3)** requirements.

R2 (Safety):

```
2025/11/12 12:23:23 [A] Requesting critical section (ts=32)
2025/11/12 12:23:23 [C] Sent REPLY(ts=34) to A
2025/11/12 12:23:23 [A] Received REPLY from C (1/2)
2025/11/12 12:23:26 [B] Sent REPLY(ts=35) to A
2025/11/12 12:23:24 [A] Received REPLY from B (2/2)
2025/11/12 12:23:24 [A] ENTERING critical section
2025/11/12 12:23:26 [A] EXITING critical section
```

Because the nodes run on independent local clocks, the wall-clock timestamps in the logs are not perfectly aligned. For instance, as we see Node A's log may show a "Received REPLY" event before the corresponding "Sent REPLY" appears in another node's log. However, the Lamport timestamps (ts=32, ts=34, ts=35) reflect the correct logical ordering of events. These logical clocks, implemented in **LamportClock.Tick()** and **Receive()**, ensure causal consistency across nodes.

R3 (Liveliness):

The logs also show that deferred requests are not lost — they are eventually served once the current holder exits.
For instance, when **C** finishes and releases its deferred replies:

```
2025/11/12 12:23:20 [B] Requesting critical section (ts=29)
2025/11/12 12:23:20 [C] Deferring reply to B
2025/11/12 12:23:22 [C] EXITING critical section
2025/11/12 12:23:23 [C] status: inCS=false, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:24 [C] Sent REPLY(ts=32) to B
```

Immediately after, **B** receives this reply and enters its own critical section:

```
2025/11/12 12:23:20 [B] Received REPLY from A (1/2)
2025/11/12 12:23:22 [B] Received REPLY from C (2/2)
2025/11/12 12:23:22 [B] ENTERING critical section
```

This illustrates that deferred replies are released in **releaseDeferredReplies()** after a node exits, ensuring that all waiting nodes eventually proceed.

Appendix – Systems Log

Node A

```
go run ./cmd/node/ -id A -port 5000 -peers A=127.0.0.1:5000,B=127.0.0.1:5001,C=127.0.0.1:5002
2025/11/12 12:22:52 [A] Listening on port 5000
2025/11/12 12:22:52 [A] Server is ready.
2025/11/12 12:22:53 [A] Connected to peer B (127.0.0.1:5001)
2025/11/12 12:22:53 [A] Connected to peer C (127.0.0.1:5002)
2025/11/12 12:22:53 [A] Connected to all peers.
2025/11/12 12:23:01 [A] Requesting critical section (ts=4)
2025/11/12 12:23:01 [A] Received REPLY from C (1/2)
2025/11/12 12:23:01 [A] Sent REPLY(ts=3) to B
2025/11/12 12:23:01 [A] Received REPLY from B (2/2)
2025/11/12 12:23:01 [A] ENTERING critical section
2025/11/12 12:23:03 [A] status: inCS=true, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:03 [A] EXITING critical section
2025/11/12 12:23:05 [A] Sent REPLY(ts=10) to C
2025/11/12 12:23:09 [A] Requesting critical section (ts=13)
2025/11/12 12:23:09 [A] Received REPLY from C (1/2)
2025/11/12 12:23:09 [A] Sent REPLY(ts=12) to B
2025/11/12 12:23:09 [A] Received REPLY from B (2/2)
2025/11/12 12:23:09 [A] ENTERING critical section
2025/11/12 12:23:11 [A] Deferring reply to C
2025/11/12 12:23:11 [A] EXITING critical section
2025/11/12 12:23:13 [A] status: inCS=false, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:13 [A] Sent REPLY(ts=22) to B
2025/11/12 12:23:13 [A] Sent REPLY(ts=20) to C
2025/11/12 12:23:15 [A] Requesting critical section (ts=23)
2025/11/12 12:23:15 [A] Received REPLY from C (1/2)
2025/11/12 12:23:15 [A] Received REPLY from B (2/2)
2025/11/12 12:23:15 [A] ENTERING critical section
2025/11/12 12:23:17 [A] EXITING critical section
2025/11/12 12:23:20 [A] Sent REPLY(ts=29) to C
2025/11/12 12:23:20 [A] Sent REPLY(ts=31) to B
2025/11/12 12:23:23 [A] status: inCS=false, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:23 [A] Requesting critical section (ts=32)
2025/11/12 12:23:23 [A] Received REPLY from C (1/2)
2025/11/12 12:23:24 [A] Received REPLY from B (2/2)
2025/11/12 12:23:24 [A] ENTERING critical section
2025/11/12 12:23:26 [A] EXITING critical section
exit status 0xc000013a
```

Manual Exit in terminal

Node B

```
go run ./cmd/node/ -id B -port 5001 -peers A=127.0.0.1:5000,B=127.0.0.1:5001,C=127.0.0.1:5002
2025/11/12 12:22:52 [B] Listening on port 5001
2025/11/12 12:22:52 [B] Server is ready.
```

```
2025/11/12 12:22:52 [B] Connected to peer A (127.0.0.1:5000)
2025/11/12 12:22:52 [B] Connected to peer C (127.0.0.1:5002)
2025/11/12 12:22:52 [B] Connected to all peers.
2025/11/12 12:22:59 [B] Requesting critical section (ts=1)
2025/11/12 12:22:59 [B] Received REPLY from A (1/2)
2025/11/12 12:22:59 [B] ENTERING critical section
2025/11/12 12:22:59 [B] Received REPLY from C (2/2)
2025/11/12 12:23:01 [B] Deferring reply to A
2025/11/12 12:23:01 [B] EXITING critical section
2025/11/12 12:23:02 [B] status: inCS=false, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:03 [B] Sent REPLY(ts=7) to A
2025/11/12 12:23:07 [B] Sent REPLY(ts=9) to C
2025/11/12 12:23:07 [B] Requesting critical section (ts=10)
2025/11/12 12:23:07 [B] Received REPLY from C (1/2)
2025/11/12 12:23:07 [B] Received REPLY from A (2/2)
2025/11/12 12:23:07 [B] ENTERING critical section
2025/11/12 12:23:09 [B] Deferring reply to A
2025/11/12 12:23:09 [B] EXITING critical section
2025/11/12 12:23:11 [B] Sent REPLY(ts=19) to C
2025/11/12 12:23:11 [B] Sent REPLY(ts=17) to A
2025/11/12 12:23:12 [B] status: inCS=false, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:13 [B] Requesting critical section (ts=20)
2025/11/12 12:23:13 [B] Received REPLY from A (1/2)
2025/11/12 12:23:13 [B] Received REPLY from C (2/2)
2025/11/12 12:23:13 [B] ENTERING critical section
2025/11/12 12:23:15 [B] Deferring reply to A
2025/11/12 12:23:15 [B] EXITING critical section
2025/11/12 12:23:17 [B] Sent REPLY(ts=26) to A
2025/11/12 12:23:20 [B] Requesting critical section (ts=29)
2025/11/12 12:23:20 [B] Received REPLY from A (1/2)
2025/11/12 12:23:22 [B] Sent REPLY(ts=28) to C
2025/11/12 12:23:22 [B] Received REPLY from C (2/2)
2025/11/12 12:23:22 [B] ENTERING critical section
2025/11/12 12:23:22 [B] status: inCS=true, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:23 [B] Deferring reply to A
2025/11/12 12:23:24 [B] EXITING critical section
2025/11/12 12:23:26 [B] Sent REPLY(ts=35) to A
Exit status 0xc000013a
```

Manual Exit in terminal

Node C

```
go run ./cmd/node/ -id C -port 5002 -peers A=127.0.0.1:5000,B=127.0.0.1:5001,C=127.0.0.1:5002
```

```
2025/11/12 12:22:52 [C] Listening on port 5002
2025/11/12 12:22:52 [C] Server is ready.
2025/11/12 12:22:52 [C] Connected to peer A (127.0.0.1:5000)
2025/11/12 12:22:53 [C] Connected to peer B (127.0.0.1:5001)
2025/11/12 12:22:53 [C] Connected to all peers.
2025/11/12 12:22:59 [C] Sent REPLY(ts=3) to B
2025/11/12 12:23:01 [C] Sent REPLY(ts=6) to A
2025/11/12 12:23:03 [C] status: inCS=false, requesting=false, replyCount=0/2, deferred=map[]
```

```
2025/11/12 12:23:05 [C] Requesting critical section (ts=7)
2025/11/12 12:23:05 [C] Received REPLY from B (1/2)
2025/11/12 12:23:05 [C] ENTERING critical section
2025/11/12 12:23:05 [C] Received REPLY from A (2/2)
2025/11/12 12:23:07 [C] EXITING critical section
2025/11/12 12:23:07 [C] Sent REPLY(ts=13) to B
2025/11/12 12:23:09 [C] Sent REPLY(ts=15) to A
2025/11/12 12:23:11 [C] Requesting critical section (ts=16)
2025/11/12 12:23:11 [C] Received REPLY from B (1/2)
2025/11/12 12:23:11 [C] Received REPLY from A (2/2)
2025/11/12 12:23:11 [C] ENTERING critical section
2025/11/12 12:23:13 [C] status: inCS=true, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:13 [C] Deferring reply to B
2025/11/12 12:23:13 [C] EXITING critical section
2025/11/12 12:23:15 [C] Sent REPLY(ts=25) to A
2025/11/12 12:23:15 [C] Sent REPLY(ts=23) to B
2025/11/12 12:23:20 [C] Requesting critical section (ts=26)
2025/11/12 12:23:20 [C] Received REPLY from B (1/2)
2025/11/12 12:23:20 [C] ENTERING critical section
2025/11/12 12:23:20 [C] Received REPLY from A (2/2)
2025/11/12 12:23:20 [C] Deferring reply to B
2025/11/12 12:23:22 [C] EXITING critical section
2025/11/12 12:23:23 [C] Sent REPLY(ts=34) to A
2025/11/12 12:23:23 [C] status: inCS=false, requesting=false, replyCount=2/2, deferred=map[]
2025/11/12 12:23:24 [C] Sent REPLY(ts=32) to B
exit status 0xc000013a
```