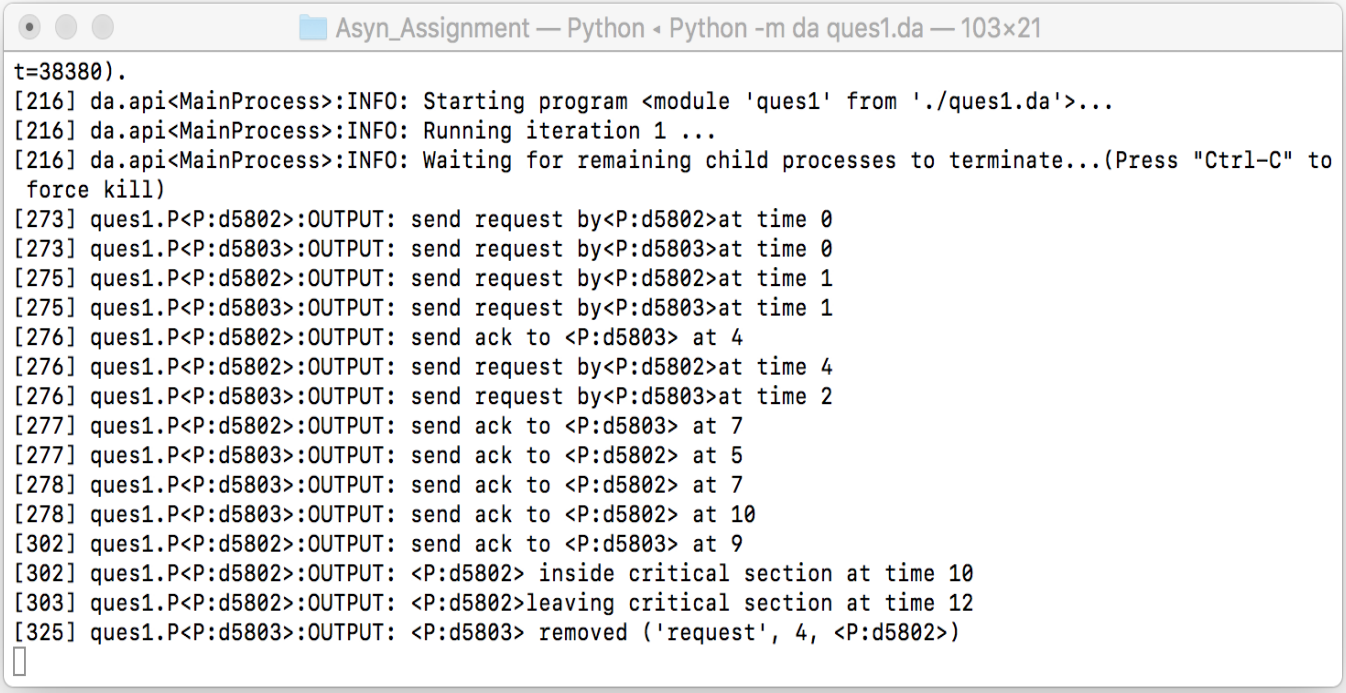
Ans 2.




1) **Liveliness Violation**: Liveliness means that the system should make progress despite the fact that the concurrent components are executing.

   **Let's consider a case of liveliness violation:**

   In the above execution trace, I have 2 processes and 3 requests.
   Let' say <P:d5803> denotes P3 process and <P:d5802> denotes P2 process.

   P3 process sends requests at time 0, 1, 2 and P2 process sends requests at time 0, 1, 4.

   P2 enters critical section at time = 10 and left critical section at time = 12.

   So at time = 12:
   Request Queue (P2) = ((P2, 0), (P3, 0), (P2,1), (P3, 1), (P3,2), (P2,4))
   Request Queue (P3) = ((P2, 0), (P3, 0), (P2,1), (P3, 1), (P3,2), (P2,4))

When P2 exits critical section it sends release resource to P3. Then P3 removes **any of the** request of P2 from its queue. So it removes request of logical clock value = 4 as evident from the last statement of snapshot.
Now in queue of P3 it contains 2 requests of lowest timestamp ((P2, 0), (P3, 0)). But since P2 has smaller process Id than P3 so it will wait for the P2 to finish that request first.
In queue of P2 it contains lowest value as (P3,0) so it will wait for the P3 to finish that request first.
So both the processes will wait for each other to finish their tasks first leading to deadlock, thus violating the liveliness of the system.

Solution: In this case Processes should not remove any of the timestamps from its queue, it should remove that requests which have lower timestamp.

2) **Safety Violation:** Safety means that something bad will never happen in the distributed system. To check for safety conditions no two processes should be in the critical section at the same time. If at any time 2 processes are in the critical section that means safety is violated. Also, the operating system maintains a CS lock which is incremented whenever a process enters the critical section and decremented whenever the process exits from the critical section. So if at any time CS lock is greater than 1 this implies that 2 processes entered the critical section at the same time and safety is violated.

**Let's consider the case of safety violation:**
We have 2 processes P0 and P1.
P0 sends requests resource messages at time t = 0, 2, 4.
P1 sends request resource messages at time t = 1.

P0 queue will look like: ((P0, 0), (P1, 1), (P0, 2), (P0, 4))
P1 queue will look like: ((P0, 0), (P1, 1), (P0, 2), (P0, 4))

Since P0 request has the smaller clock value so it will enter into the critical section first. After finishing its execution, P0 removed its request of (P0, 4) if it can remove **any of** its request from the queue. And after release resource message from P0, P1 removed (P0, 0) request from its queue.

Now,
P0 queue will look like: ((P0, 0), (P1, 1), (P0, 2))
P1 queue will look like: ((P1, 1), (P0, 2), (P0, 4))

Since P0 queue has smaller timestamp of (P0, 0), so it will enter the critical section. Also P1 queue has smaller timestamp of (P1, 1) at its front so P1 will also enter critical section, thus violating the safety property of distributed system.

Solution: When a process is releasing the source it should only remove the timestamp from its queue which has the smallest logical clock value. Also when other processes are getting the release resource message from a process, they should remove the minimum value request logical clock message of that process.