Exercise A.

1.1 Question 1: Which of these times should it use to set its clock?

The client should choose the time with the smallest Round-trip time (RTT) which is 20 ms

1.2 Question 2: To what time should it set it?

By assuming that the client's reate of clock drift is sufficiently small, and that the elapsed time is split equally before and after S placed t in m_t we can use that formular $t + T_{round}/2$
Where t is the time value inserted in a message m_t at the last possible point before transmission from the server, T_{round} is the total round-trip time taken to send a request m_r and receive the reply m_t.
We then have that the client should set its time to

$$10:54:28:342(hr:min:sec:ms) + 0.02(s)/2$$

which is 10:54:28:352 (hr:min:sec:ms)

1.3 Question 3: Estimate the accuracy of the setting with respect to the server's clock

The accuracy can be calculated using the formular

$$\pm(T_{round}/2 - min)$$

where min is the *minimum transmission delay* and T_{round} is the minimum RTT value. The minimum transmission delay is at least 0 ms, since it cannot be faster than instant and so we get an accuracy of

$$\pm(20ms/2 - 0ms) = \pm 10ms$$

1.4 Question 4: If it is known that the time between sending and receiving a message in the system concerned is at least 8 ms, do your answers change?

If we know that the minimum message transfer time is 8ms, then the setting stays the same but the accuracy improves to

$$\pm(20ms/2 - 8ms) = \pm 2ms$$

Exercise B.

An NTP server B receives server A's message at 16:34:23.480 bearing a timestamp 16:34:13.430 and replies to it. A receives the message at 16:34:15.725, bearing B's timestamp 16:34:25.7. Estimate the offset between B and A, and the accuracy of the estimate.

The estimated offset is:

$$\frac{(23{,}480 - 13{,}430) + (25{,}7 - 15{,}725)}{2} = 10{,}0125\ seconds$$

And the accuracy of the estimate is:

$$(15{,}725 - 23{,}480) - (25{,}700 - 23{,}480) = 0.075\ seconds$$

Exercise C.

We have the following logical ordering:

p1 and q1 and r1 happens concurrently.

p1 --> p2

p2 --> p3

p3 --> p4

q1 --> q2

q2 --> q3

q3 --> q4

q4 --> q5

q5 --> q6

q6 --> q7

r1 --> r2

r2 --> r3

r3 --> r4

p1 --> q2

q1 --> p2

q4 --> r3

q5 --> p4

q1 --> r4

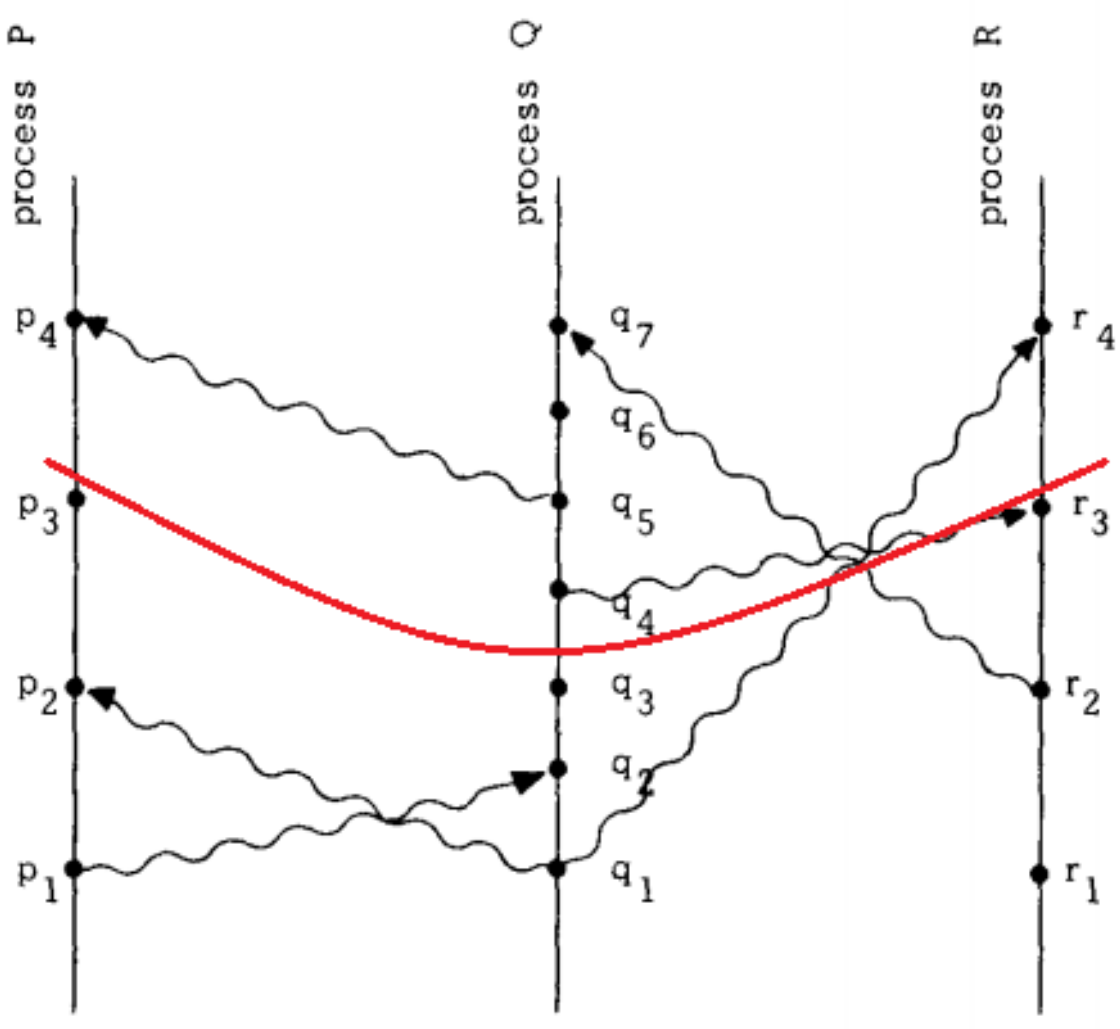
r2 --> q7

(2) ***Identify 4 consistent cuts.***

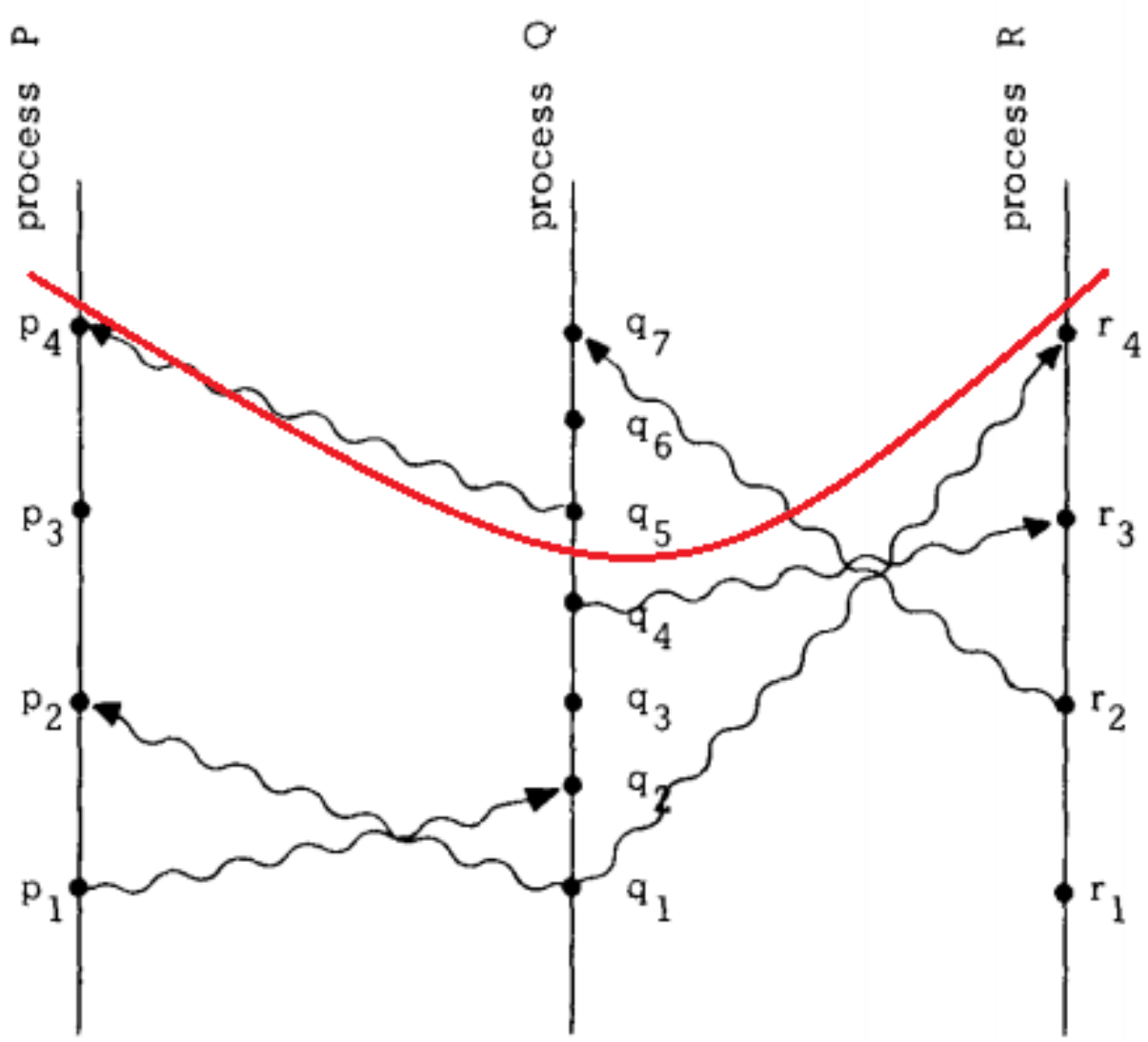
- (t1,q1,r1)
- (p3,q4,r3)
- (p2,q2,r2)
- (p4,q7,r4)

(3) ***Identify 4 inconsistent cuts.***

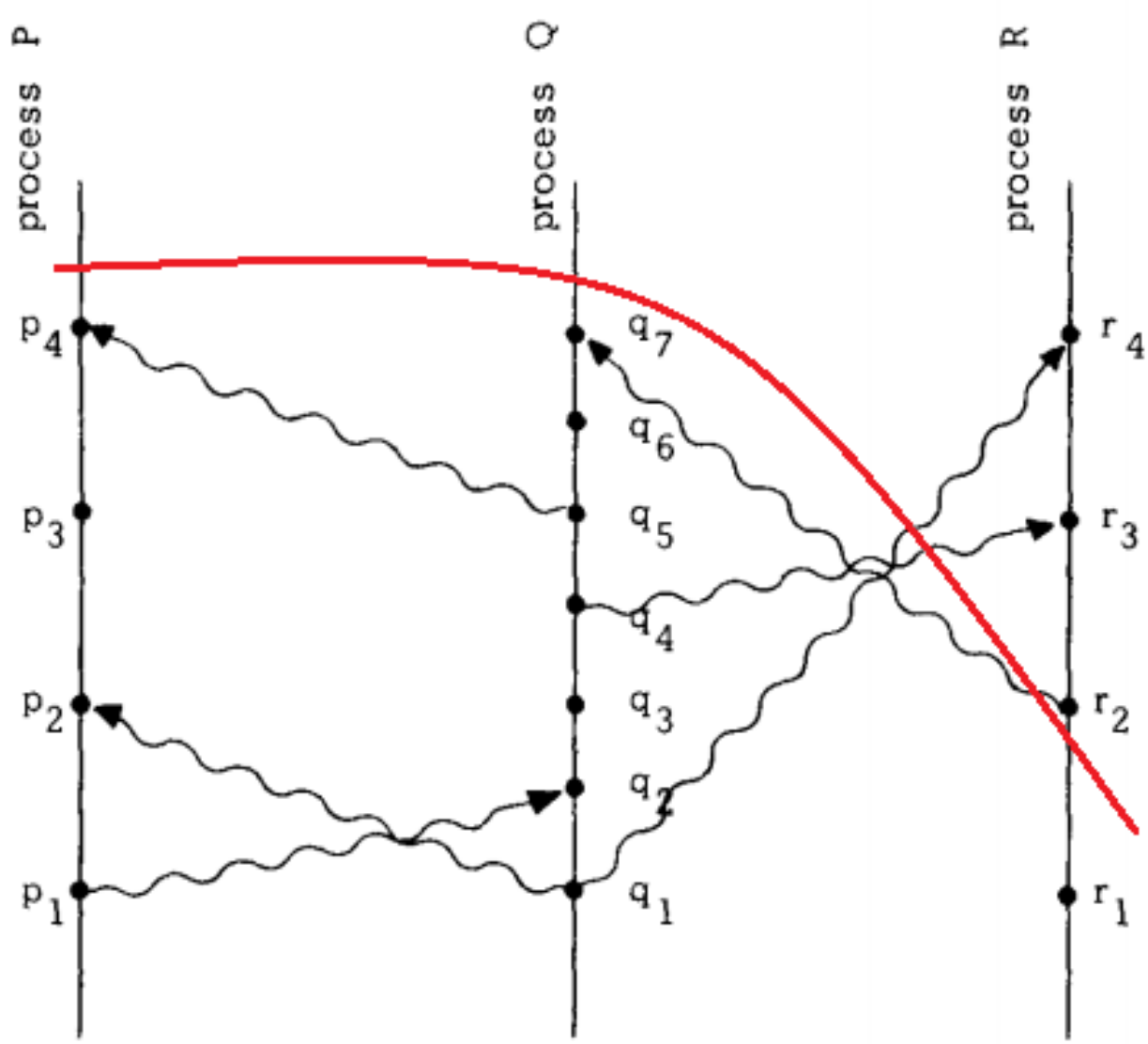
- (p3,q3,r3) - Since r3 can´t be instanited / activated before q4.



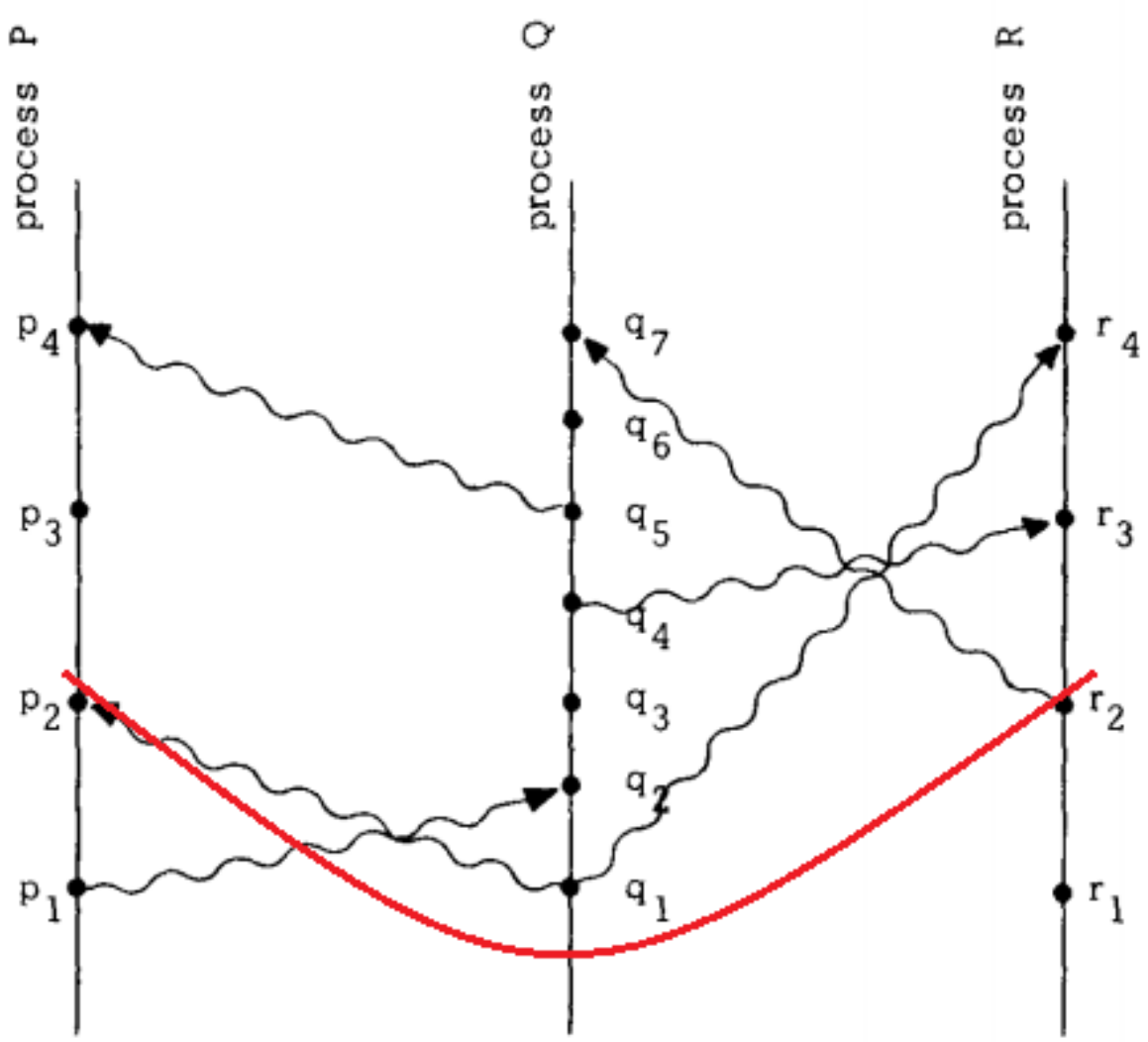
- (p4,q4,r4) - Since p4 can´t be instanited / activated before q5.



- (p4,q7,r2) - Since q7 can´t be instanited / activated before r2.



- (p2,q0,r2) - Since p2 can´t be instanited / activated before q1.



(4) ***Write 2 different linearisations of the events in this diagram.***

Version 1:

(p1,q1,r1,p2,q2,r2,p3,q3,q4,r3,q5,p4,q6,r4,q7)

Version 2:

(r1,r2,p1,q1,q2,q3,q4,q5,q6,q7,p2,p3,p4,r3,r4)

Exercise D.

1. If k is odd for all processes and no messages are pending, nothing will happen and the system has reached a deadlock. The initial state may also be a single odd number k for all processes.
2. If k is odd for all processes nothing will happen and the system has reached a deadlock.
3. We wish to achieve some notion of a global state of a system when using the snapchot algorithm. This requires to inspect all systems simultaneously (exactly at some point), which is not possible. We can get fragments (snapshots/cuts) that can be reasonably considered as global states (things happening at the same time). The snapshot algorithm is used to compute these fragmented parts of the system by using marker messages that tell when processes have recorded their own state and messages have been delivered.
4. Has a deadlock occurred (We assume a process can receive markers and open channels even if it has a state which is odd)
    a. A marker is sent to the first process p1
    b. On response is received from p1 and the state 3 is stored and a channel is opened to p1 listening to p1 state
    c. A marker is sent to process p2
    d. A response is received from p2 and the state 4 is stored and a channel is opened to p2 listening listening to its state
    e. A marker is sent to process p3
    f. A response is received from p3 and the state 7 is stored and a channel is opened to p3 listening listening to its state
    g. The snapshot algorithm has a open channel to each process and can now conclude if a deadlock has occurred. Process p2 has still an active sending channel and therefore no deadlock has occurred

Initial global states 3, 5, 7 and no messages in transit. Will the snapchot conclude a deadlock?

a. Process Pi(3). Process pj(5) and pk(7) do not record their own k state because it is odd and no messages are pending.
b. Process Pi creates a special "Marker" message.
c. Process Pi sends out the marker message on each channel.
d. Process Pi starts recording incoming messages on each incoming channel.
e. Markers are sent out from all processes but their states are not randomly updated since k is odd.
f. The snapshot algorithm terminates since all processes have received a marker.

Conclusion: A deadlock occurs since all processes have an odd k state value. To avoid a deadlock, at least one process should have an even k state value to ensure that all processes eventually reach an even k state value (due to markers), after which all states have been recorded by the snapshot algorithm.