Homework 5

Mohan Srinivas Naga Satya Pothukuchi

NUID: 001821918

Exercise 14.2

A clock is reading 10:27:54.0 (hr:min:sec) when it is discovered to be 4 seconds fast. Explain why it is undesirable to set it back to the right time at that point and show (numerically) how it should be adjusted so as to be correct after 8 seconds have elapsed.

In distributed systems, it is necessary that the two clocks maintain synchronization to measure the occurrence of processes within the system. In an instance when the clock is not synchronized, it is not desirable that the clock be set back in time due to drift that each clock maintain. In a software system, the drift of each clock defines the measure of each update. So, maintaining the drift between two clocks is raises arbitrary failures within the system.

For example, the make tool in a system executes the files based on the time they are changed. In that case, when the clock is set back in time the file's modification changes. Though make tool doesn't erroneously compile the files, other tools which are dependent on the clock might be affected.

In a software machine, the monotonicity can be achieved by changing the rate at which the updates are being performed. This can be mathematically calculated as:

=> $S = aH + b$

Where S = clock's software time.
H = hardware clock time.

As there is a drift, the original time	t = 10:27:50
As stated, the correct time will be	H = 10:27:58
Also stated, the software time,	S = 10:27: 54

Hence, $t+4 = a(t+8 - t) + t$

$a = 0.5$

So, the software clock update rate must 0.5 per tick.

Exercise 14.4

A client attempts to synchronize with a time server. It records the following round-trip times and timestamps returned by the server: Which of these times should it use to set its clock? To what time should it set it? Estimate the accuracy of the setting with respect to the server's clock. If it is known that the time between sending and receiving a message in the system concerned is at least 8 milliseconds, do your answers change?

Round-trip (ms)	*Time (hr:min:sec)*
22	*10:54:23.674*
25	*10:54:25.450*
20	*10:54:28.342*

In principle, the receiving process could set its clock to the time t + T_{trans} ,
where T_{trans} is the time taken to transmit m between them.

The two clocks would then agree since the aim is internal synchronization. But, as other processes are competing for resources with the processes to be synchronized at their respective nodes, and other messages compete with m for the network resources, a minimum time can be obtained.
Where t_{min} = 8ms

Let the uncertainty in the message transmission time be u, so that *u = (max – min).*

- If the receiver sets its clock to be *t + min* , then the clock skew may be as much as u, since the message may in fact have taken time max to arrive.
- Similarly, if it sets its clock to *t + max* , the skew may again be as large as u.
- If however, it sets its clock to the halfway point, *t + (max + min) / 2* , then the skew is at most *u / 2* .

The system should it's time *t* = 10:54:28:342
This is because, the halfway point t/2 = t + (20 + 8) / 2
= t + 0.16s
Where as the skew = (20-8) /2
= 12/2
= 0.06s
As the skew is least for the above value, *the correct time would be t = 10:54:28:342*

Question 3:

Assigning Lamport Timestamps to processes P_1, P_2, P_3, P_4

- P_1 sends a message to P_3 (to event e).
  - ➔ Let, $L(P_1) = 0$
  - ➔ Before sending the message, $L(P_1) = L(P_1) + 1$
    $L(P_1) = 1$
    $t = L(P_1)$
  - ➔ Sends the message as (m, t)
  - ➔ Before receiving message on P3, $L(P_3) = 0$
  - ➔ $L(P_3) = MAX(L(P_3), t)$
    $L(P_3) = 1$
    $L(P_3) = L(P_3) + 1$
    $L(P_3) = 2$
- P_1 receives a message from P_3 (from event g).
  - ➔ Before performing *send(m),* $L(P_3) = L(P_3) + 1$
    $L(P_3) = 3$
  - ➔ Perform *send(m, $L(P_3)$)* to P_1
  - ➔ Before P_1 performing *receive(m),* $L(P_1) = 1$
  - ➔ $L(P_1) = Max(L(P_3), L(P_1))$
    $L(P_1) = 3$
    $L(P_1) = L(P_1) + 1$
    $L(P_1) = 4$
- P_2 executes a local event.
  - ➔ Before performing the event, $L(P_2) = 0$
  - ➔ $L(P_2) = L(P_2) + 1$
  - ➔ $L(P_2) = 2$
- P_2 receives a message from P_3 (from event f).
  - ➔ Before performing send(m), $L(P_3) = 3$
    $L(P_3) = L(P_3) + 1$
    $L(P_3) = 4$
  - ➔ Perform send(m, $L(P_3)$) to P_3
  - ➔ Before P_2 performing *receive(m),* $L(P_3) = 1$
  - ➔ $L(P_3) = Max(L(P_3), L(P_2))$
    $L(P_3) = 4$
    $L(P_3) = L(P_3) + 1$
    $L(P_3) = 4$
- P_3 receives a message from P_1 (from event a).
  - ➔ Before performing send(m), $L(P_1) = 4$
    $L(P_1) = L(P_1) + 1$
    $L(P_1) = 5$
  - ➔ Perform send(m, $L(P_1)$) to P_3
  - ➔ Before P_3 performing *receive(m),* $L(P_3) = 4$
  - ➔ $L(P_3) = Max(L(P_1), L(P_3))$
    $L(P_3) = 5$

$L(P_3) = L(P_3) + 1$
$L(P_3) = 6$

- P_3 sends a message to P_2 (to event d).
  - ➔ Before sending the message, $L(P_3) = L(P_3) + 1$
    $L(P_3) = 7$
    $t = L(P_3)$
  - ➔ Sends the message as (m, t)
  - ➔ Before receiving message on P_2, $L(P_2) = 4$
  - ➔ $L(P_2) = MAX(L(P_2), t)$
    $L(P_2) = 7$
    $L(P_2) = L(P_2) + 1$
    $L(P_2) = 8$
- P_3 sends a message to P_1 (to event b).
  - ➔ Before sending the message, $L(P_3) = L(P_3) + 1$
    $L(P_3) = 8$
    $t = L(P_3)$
  - ➔ Sends the message as (m, t)
  - ➔ Before receiving message on P_1, $L(P_1) = 4$
  - ➔ $L(P_1) = MAX(L(P_1), t)$
    $L(P_1) = 8$
    $L(P_1) = L(P_1) + 1$
    $L(P_1) = 9$
- P_4 executes a local event.
  - ➔ Before performing the event, $L(P_4) = 0$
  - ➔ $L(P_4) = L(P_4) + 1$
  - ➔ $L(P_4) = 1$

Lamport TimeStamps (P1, P2, P3, P4) = (9, 8, 8, 1)

Assigning vector timestamps for processes P_1, P_2, P_3, P_4

- P_1 sends a message to P_3 (to event e).
  - ➔ (1, 0, 1, 0)
- P_1 receives a message from P_3 (from event g).
  - ➔ (2, 0, 2, 0)
- P_2 executes a local event.
  - ➔ (2, 1, 2, 0)
- P_2 receives a message from P_3 (from event f).
  - ➔ (2, 2, 3, 0)
- P_3 receives a message from P_1 (from event a).
  - ➔ (3, 2, 4, 0)
- P_3 sends a message to P_2 (to event d).
  - ➔ (3, 3, 5, 0)
- P_3 sends a message to P_1 (to event b).
  - ➔ (4, 3, 6, 0)
- P_4 executes a local event.
  - ➔ (4, 3, 6, 1)

Which events are concurrent with event *d*?

As events C and H are not accurately represented, either of them could have possibly occurred.

- ➔ If event C occurred in P2, then event C occurred concurrently with D
- ➔ If event H occurend in P2, then event H occurred concurrently with D

Question 4

You are synchronizing your clock from a time server using Cristian's algorithm and observe the following times:

- timestamp at client when the message leaves the client: 6:22:15.100
- timestamp generated by the server: 6:21:10.700
- timestamp at client when the message is received at client: 6:22:15.250

Q1. To what value do you set the client's clock?

Q2. If the best-case round-trip message transit time is 124 msec (0.124 sec), what is the error of the clock on the client?

Q1) According to Cristian's algorithm,

A process p should set its clock is $t + T_{round} / 2$

Hence, t = 6:22:15.100

T_{round} = ($Client_{receive}$ – $Client_{send}$)

= 6:22:15.100 – 6:22:15.250

= 150ms

Client clock = (6:22:15.100 + 75ms)

= 6:22:15.175

Q2)

The earliest point at which server could have placed the time in m_t was *min* after client dispatched m_r. The earliest point at which S could have placed the time in m_t was min after p dispatched m_r . The latest point at which it could have done this was min before m_t arrived at p. The time by S's clock when the reply message arrives is therefore in the range [t + min, t + Tround – min] .

here, min = 124ms

t = 6:22:15.100

T_{round} = 150ms

Range = [6:22:15.126, 6:22:15.224]

Client received = 6:22:15.250

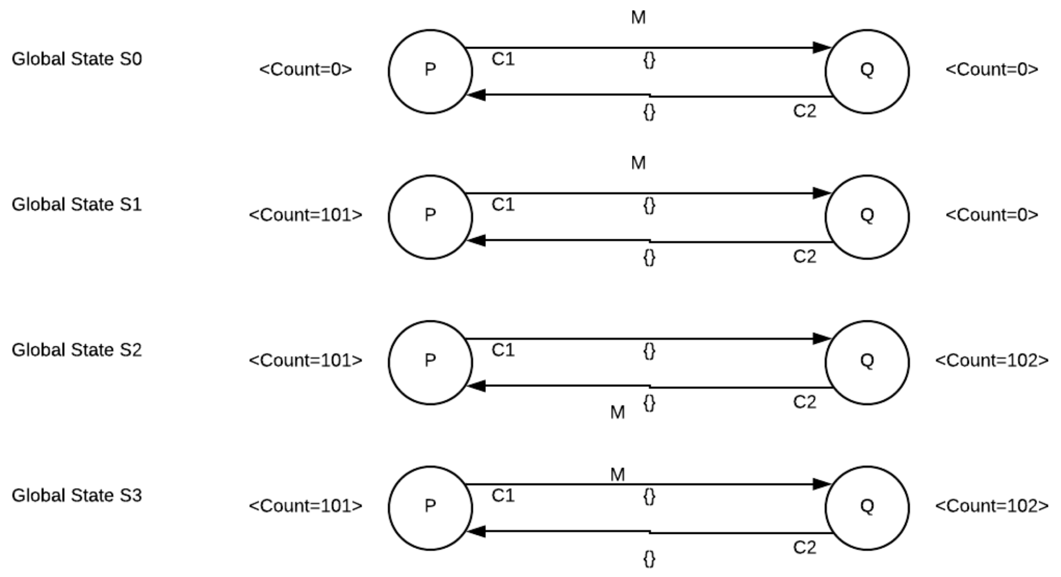
Error = $Client_{received}$ - Max(Range)

= 6:22:15.250 – 6:22:15:224

= 26ms.

Exercise 14.4

Two processes P and Q are connected in a ring using two channels, and they constantly rotate a message m. At any one time, there is only one copy of m in the system. Each process's state consists of the number of times it has received m, and P sends m first. At a certain point, P has the message and its state is 101. Immediately after sending m, P initiates the snapshot algorithm. Explain the operation of the algorithm in this case, giving the possible global state(s) reported by it.



- P sends message m
- P records state 101
- Initiates the snapshot algorithm
- P sends marker M to Q, and the marker receiving rule obligates a process that has not recorded its state to do so, making Q state 102
- The state of the channel from P to Q as {}
- *The marker sending rule obligates processes to send a marker after they have recorded their state, but before they send any other messages.*
- Q sends marker M as per marker sending rule
- P receives marker M
- P records the state of the channel from Q as set of messages received.