cha1. You MUST answer this question.

(a) Under which condition(s) are two events e and e' called concurrent events? [2 marks]

For example two event may be deemed concurrent meaning that we do not know which occurred first, but we will never erroneously ascertain that e occurred before e'.

The events e and e' may be considered concurrent if we do not know which occurred first, explicitly, we cannot say $e \rightarrow e'$ or $e' \rightarrow e$.

(b) Consider a cryptographic security protocol for authenticated communication between two parties Alice and Bob, via an authentication server called Sara. Explain the concept of a challenge in such cryptographic security protocols. [6 marks]

Answer 1

bob and Alice Both trust sarah.

if bob wants to send a message to alice he must be able to prove his identity to alice.

He can do this if alice trust sarah, by getting sarah to sign sign his public key with her private key.

Alice can then verify that bob is who he says he is by checking that his public key has been signed by sarah whose signature she trust.

So when bob transmits a message he must send the signed message from Alice proving his key is valid and his message signed with his key. If alice trust the key because she trusts the message from sarah she will then trust the message from bob.

This is a chain of trust. Using a public key infrastructure

Answer 2

A challenge is a process done by the authentication server Sara, which issues an unknown challenge value to the user, this is then algorithmically combined with their credentials to verify their identity. This aspect of the authentication process is to try and stop a replay attack, the idea is that a recorded but valid authentication would be unique to the challenge that was issued and could not be used unless the same challenge was re-issued.

Answer 3

The purpose of a challenge in security protocols is ultimately to prevent a replay attack.

In the simplified version of this protocol, Alice sends Sarah a request for a ticket (containing a shared key, and Alice's identity) to send to Bob. If an attacker captures this ticket as it is sent to Bob, they could impersonate Alice and send the ticket to Bob again in the future. **This would be successful if the attacker somehow found out K_{AB} prior to sending the ticket to Bob.**

A challenge requires the 'prover' combining a received value with their authentication to create something that Sarah/Bob can verify, ensuring that the 'prover' currently has the key and is not simply replaying earlier messages.

(c) Suggest how to adapt the Bully algorithm for leader election to deal with temporary network partitions (slow communication) and slow processes. [9 marks]

http://courses.engr.illinois.edu/cs425/fa2011/hw/hw2-sol.pdf

//Answer: In case of network partitions, subgroups will be formed. Each subgroup can run Bully algorithm and elect a coordinator with the highest ID in the subgroup. When the network heals, the subgroups should merge and elect one coordinator with the highest ID.

\zThe bully algorithm could be adopted by allowing networks to partition and having separate leaders elected in each of the partition, but when the networks reconnect the bully algorithm could detect that there were multiple Elected processes then when the one with higher priority would automatically call a new election bullying the lower one into giving up its position

WIKIPEDIA says

Note that if P receives a victory message from a process with a lower ID number, it immediately initiates a new election. This is how the algorithm gets its name - a process with a higher ID number will bully a lower ID process out of the coordinator position as soon as it comes online.

(d) Consider the following set of vector clock values that have been observed for six different events a, b, . . . , f in an asynchronous distributed system: [8 marks]

V (a) = (4,4,2)
V (b) = (4,2,2)
V (c) = (2,2,3)
V (d) = (4,3,2)
V (e) = (4,4,4)
V (f) = (3,2,2)

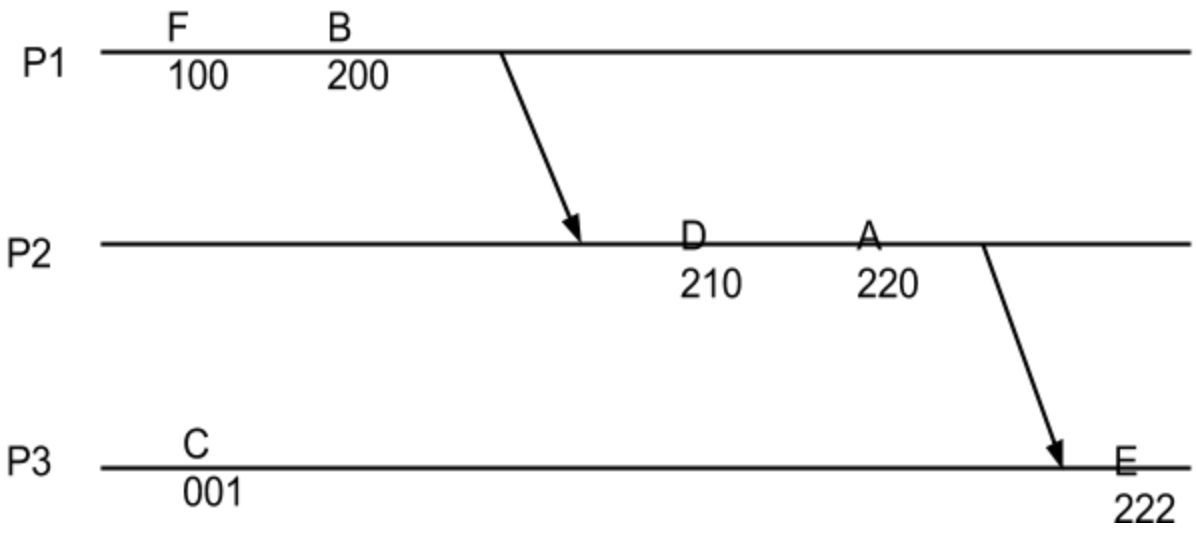
Draw a timeline diagram that illustrates the partial order happened-before relation on the events. Draw one timeline per process. Draw the events that belong to each process in the correct local order along the correct timeline. Indicate where message exchanges between processes occur.

Henrik suggests subtracting 2 from all clocks to make the numbers easier to interpret.

```
a  4  4  2
b  4  2  2
c  2  2  3   <
d  4  3  2
e  4  4  4
f  3  2  2   <
```

In general: Start with the lowest values (e.g. c or f). Events with just one incremented clock different (e.g. 4 4 2 [a] and 4 3 2 [d]) will be directly connected, either by being on the same processor or connected by a message, the later event will be on processor 2 as that's the one that's been incremented.

Example of a timeline:



Answer to question:

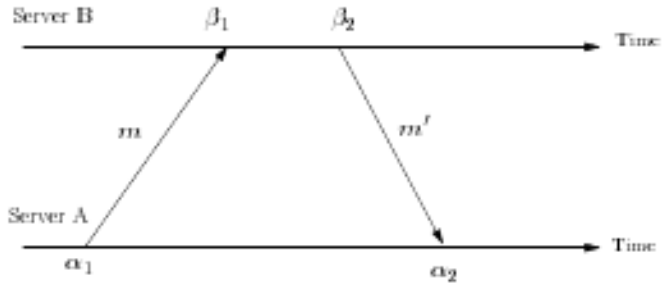
```
1  ----F(3, 2,2)----B(4,2,2)--------------------------
                          \
                            \
                             \
2  --------------------D(4, 3, 2)----A(4, 4,2)---------
                                              \
                                               \
                                                \
3 ------C(2,2,3)--------------------------E(4, 4, 4)---
```

2. (a) Which properties does a distributed system need to satisfy in order to qualify as a *synchronous* distributed system? [*6 marks*]

(b) Explain the difference between hard-mounted and soft-mounted filesystems. [*2 marks*]

(c) The NFS filesystem server does not directly provide the standard Unix file operations of 'open' and 'read'. Why? What operation does NFS use instead? [*5 marks*]

(d)



Consider the exchange of messages between two time servers A and B above. Each server has a local clock which he uses to timestamp events, i.e., α_1 (resp. α_2) is the timestamp of sending message m (resp. receiving message m') by server A's clock. Similarly, β_1 (resp. β_2) is the timestamp of receiving message m (resp. sending message m') by server B's clock.

The clock of server B is fast by an offset x relative to the clock of server A at the beginning of the exchange. (You may assume that there is no additional clock drift during the exchange.)

Additionally, we assume that every message from A to B, and vice versa, takes at least time γ.

i. Derive an estimate x' for the clock offset x in terms of $\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma$. [*6 marks*]

ii. Derive an accurate error bound for this estimate (i.e., a bound on $|x'-x|$) in terms of $\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma$. [*6 marks*]

---

2. (a)

From Slides:

* time to execute each step of a computation within a process has known lower and upper bounds

* message delivery times are bound to a known value

* each process has a clock whose drift rate from real time is bounded by a known value

Also useful, from 2013/14 slides on synchronous communication:

- Operation in rounds, in a round a node performs some computation and then sends some messages, all messages sent at the end of round x are available to recipients at the start of round x+1
- Can be implemented if message transmission time is bounded by constant, say m
- Computation times for all nodes are bounded by some constant c
- Clocks are synchronized
- Each round is set to be m+c in duration

(b) Not examinable this year

(c) Not examinable this year

(d) Slides 132-133 from the lecture notes (Pairwise synchronisation)

i)

B1 = a1 + y + x' -> x' = B1 - a1 -y

B2 = a2 - y + x' -> x' = B2 - a2 + y

2x' = (B1 - a1) - y + (B2 - a2) + y -> **x' = ((B1 - a1) + (B2 - a2))/2**

I think there may be a mistake there or I misunderstood something, according to the slides our **t = t' = y,** our **O_true = x** and we want **O_guess = x',** so the equations are:

(1) b1 = a1 + y + x

(2) a2 = b2 - x + y

---- Subtract one from the other (1) - (2) to get O_guess, that is x':----

b1 - a2 = a1 - b2 + 2x'

x' = (b1 - a1 + b2 - a2) / 2

I get the same answer , but in (2) I have different signs in front of y and x, hence I subtract not add. I believe this is also done in the slides - please correct me if I am wrong :)

ii)

From slide 132 T_{round} = (a2-a1) - (B2-B1). Thus

x' - ((a2-a1)-(B2-B1))/2 <= x <= x' + ((a2-a1)-(B2-B1))/2

I agree with the above, just out of curiosity, if we do everything that is in the slides, we get that:

b1 - a1 = x + y

b2 - a2 = x - y

as t and t' are the same in our case - no additional clock drift is assumed. Thus if

O_guess = x' = (t - t') + O_true = (y - y) + x = x **=>** x' = x

Do we have to include this outcome in the answer, or do we just stick with the bound?

New:

ta=tb+x'

beta1=alpha1+tab-x'

alphat2=beta2+tba+x'

T round=tab+tba=beta1+alpha

tab=beta1-alpha1+x'>gama

tba=alhpa2-beta2-x'>gama

gama-beta1+alpha1<x'<alph2-beta2-gamma

x'≈1/2(gama-beta1+alpha1+alph2-beta2-gamma)=...

According to James Cheney and Rik Sarkar slides here is what we get

T(i-3) = a1

T(i-2) = b1

T(i-1) = b2

T(i) = a2

From the previous answer we know that

Oguess = x' = (b1 - a1 + b2 - a2) / 2

According to his slides :

Otrue is bounded by Otrue =< Oguess + Tround

Now for this situation we need to compute Tround.

Tround = (t+t') = (timeAB + timeBA)

Now here is the catch, there seems to be two ways to measure Tround either we can take time TimeAB = gamma

and get Tround = gamma + timeBA = gamma + a2-B2

or we can have TimeAB = a2-a1 -b2 +b1

so the upper bound can either be Oguess + gamma + a2-B2

Or

Oguess + a2-a1 -b2 +b1

In the revision lecture slides they say that NTP was covered on a high level only and the technical details won't be examinable. So I believe this question doesn't apply for 2014 exams

3. (a) Define the three requirements (ME1,ME2 and ME3) for mutual exclusion algorithms in message passing systems. [3 marks]

(b) Adapt the central server algorithm for mutual exclusion to handle the crash failure of any client (in any state), assuming that the server is correct and given a reliable failure detector. Comment on whether the resulting system is fault tolerant. What would happen if a client that possesses the token is wrongly suspected to have failed? [6 marks]

(c) Give an example execution of the ring-based algorithm to show that processes are not necessarily granted entry to the critical section in happened-before order of their requests. [4 marks]

(d) In a certain system, each process typically uses a critical section many times before another process requires it. Explain why Ricart and Agrawala's multicast based mutual exclusion algorithm is inefficient for this case, and describe how to improve its performance. Does your adaptation satisfy condition ME2? [12 marks]

---

3. (a) 4 4 2

* Safety Property: At any instant, only one process can execute the critical section.

* Liveness Property: This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages which will never arrive.

* Fairness: Each process gets a fair chance to execute the CS. Fairness property generally means the CS execution requests are executed in the order of their arrival (time is determined by a logical clock) in the system.

http://www2.cs.siu.edu/~mengxia/Courses%20PPT/420/chapter10coordinationAgreement.ppt

ME1: (safety) at most one process may execute in the critical section

ME2: (liveness) Request to enter and exit the critical section eventually succeed.

ME3(ordering) One request to enter the CS happened-before another, then entry to the CS is granted in that order.

ME2 implies freedom from both deadlock and starvation. Starvation involves fairness condition. The order in which processes enter the critical section. It is not possible to use the request time to order them due to lack of global clock. So usually, we use

happen-before ordering to order message requests.

(b)

Answer 1

c

Answer 2

The central server algorithm is inherently resistant to crash failures of processes which do not have the access key and are not in the critical section. Therefore, we focus on the client with the access key or a client utilising the critical section when adapting the algorithm for fault tolerance. In the canonical version there is only one access key for the critical section, in our modified version new access keys can be generated although the central server will only grant access to the most recently issued access key. In the case that the process crashes in either of these two cases and the failure detector picks it up then we should restart the process and generate a new access key for the critical section.

The overall system is still not entirely fault tolerant as the central server itself can crash, although, it will be resilient to an arbitrary number client process crashing. If the client which is holding the access key is wrongly suspected to have failed, it may try and use it's key to try and access the critical section, however it will be denied access by the central server as it's access token will have expired. +1 Great answer

Answer 3:

Add a timer on the token and make it valid only for some time. If the client having the token fails, we can simply issue a new token when the old one expired. However, the clients need to guarantee that if their token expires while accessing the device protected by the mutex, they will free that device. This would work only in a synchronous system though and that guarantee will pretty much transform this into a real-time system. Probably the 2nd answer is better but added this for diversity.

(c) Suppose again we have two processes P1 and P2 consider the following events

* Process P1 wishes to enter the critical section but must wait for the token to reach it.

* Process P1 sends a message m to process P2.

* The token is currently between process P1 and P2 within the ring, but the message m reaches process P2 before the token.

* Process P2 after receiving message m wishes to enter the critical section

* The token reaches process P2 which uses it to enter the critical section before process P1

(d) Ricart and Agrawala's algorithm multicast-based algorithm multicasts requests, and requires reply from all other processes before entering a critical section, which is expensive when one process needs several access to the critical section before any other process requests for it. One possible solution is to change the state from HELD to TEMP instead of RELEASE when the process is done with the critical section. If it needs access again, it can change the state from TEMP to HELD without sending multicast message. To satisfy ME2 the process has to change state TEMP to RELEASE if there is any request for the critical section, i.e., its queue is not empty.