UMEÅ UNIVERSITET
Institutionen för Datavetenskap

UMEÅ UNIVERISTY
Dept of Computing Science
Unique code: 1

EXAMINATION

Course: **5DV020/Distributed Systems**

Teacher in charge: Yvonne Löwstedt/Lars Larsson

Semester: HT-10
Date: 2011-01-05
Time: 09:00–13:00

Name: _____

Personal ID number: _____

Unique code for this examination: **1**

Note!

This examination will be graded anonymously. This sheet will be removed before the teacher receives the rest of the examination. The above code must therefore be on all other pages when you submit the examination to the examination supervisory staff. **Memorize** your code since it will be used as reference when the results are published.
Furthermore,

- Write the answers on the answers on the same paper as the question (the back of the paper may also be used).
- Mark the questions you have solved with a cross on the next page.
- The solutions should be neatly written. The train of thought should be easy to follow. All non-obvious assumptions must be explicitly stated.

Till skrivningsbevakaren: Avskilj detta försättsblad och stoppa i kuvert som skickas till Yvonne Löwstedt, Datavetenskap.

UMEÅ UNIVERSITET
Institutionen för Datavetenskap

UMEÅ UNIVERISTY
Dept of Computing Science
Unique code: 1

EXAMINATION

Course: **5DV020/Distributed Systems**

Teacher in charge: Yvonne Löwstedt/Lars Larsson

Semester: HT-10

Date: 2011-01-05

Time: 09:00–13:00

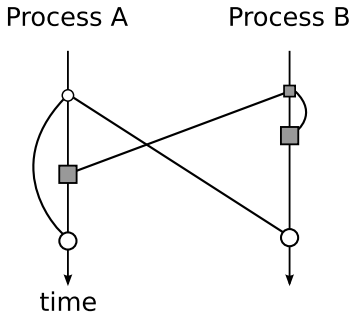
Unique code for this examination: **1**

Problem	**Solved**	**Points**
1		
2		
3		
4		
5		
6		
7		
8		
9		
Sum		
Grade		

Question 1 (2 + 2 + 2 + 2 points)

GCom requires support for several message orderings: FIFO, causal, total, and causal-total. Hybrid orderings such as the last one are interesting from many points of view, since they allow us to gain benefits of several types of orderings — without requiring much effort from us as implementers. Let us therefore devote some attention to such hybrid orderings!

For the following, assume that there are at least three processes in the system and that there is a minimum total of five messages sent between them. All processes must send at least one message (to avoid trivial solutions). You do not have to explicitly show sequencer or sequence number voting messages. Also, please provide your answer as figures with additional text (not only in writing, as the written word is open to ambiguous interpretations). The figures must be drawn in the following fashion:



A small circle denotes a send event, whereas a big circle of the same color denotes a *delivery* event. Recall the difference between receiving a message and delivering it: received messages are put in queues until they are ready to be delivered according to the chosen ordering.

a. If possible, show a message ordering that is **total**, but **not FIFO-total**. If such an ordering is impossible to show, provide a sound argument for why this is the case.

b. If possible, show a message ordering that is **causal**, but **not FIFO**. If such an ordering is impossible to show, provide a sound argument for why this is the case.

c. If possible, show a message ordering that is **FIFO**, but **not causal-total**. If such an ordering is impossible to show, provide a sound argument for why this is the case.

d. If possible, show a message ordering that is **unordered**, and **neither** FIFO, causally, nor totally ordered. If such an ordering is impossible to show, provide a sound argument for why this is the case.

Feel free to use this page as well...

. . . and this one.

Question 2 (3 × 0.5 + 4 points)

Let's consider reliable multicast for a while: the basic algorithm you implemented in the assignment sends $\mathcal{O}(n^2)$ messages for each data message, which is incredibly wasteful. It does, however, get the job done — it is a correct algorithm for implementing reliable multicast. How do we know that it is correct? Because it fulfills the three important properties of *integrity*, *validity*, and *agreement*.

a. For 0.5 pts each, what do the three properties (*integrity*, *validity*, and *agreement*) mean?

b. The reason the basic algorithm is horribly inefficient is that it makes use of unicast (point to point communication). If, however, we have access to IP multicast-enabled hardware and systems, we should be in much better shape! The book gives a sketch for how to implement *reliable* multicast using (unreliable) IP multicast. Explain how this works!

Perhaps you need this page, perhaps not. Here it is, anyway!

Question 3 (3 + 2 + 1 points)

Clock synchronization is important: many algorithms depend on there being some notion of what time it is. Examples of this is range from e.g. relatively complicated Kerberos tickets to debugging to even the cookies in your web browser. This time, we want to test your knowledge on Cristian's method for time synchronization, so let's get to it (no time to lose, eh?).

a. Describe how Cristian's method works *and* briefly discuss when and why it is applicable. Figures are good for this, but the figure will need some explanation in text as well.

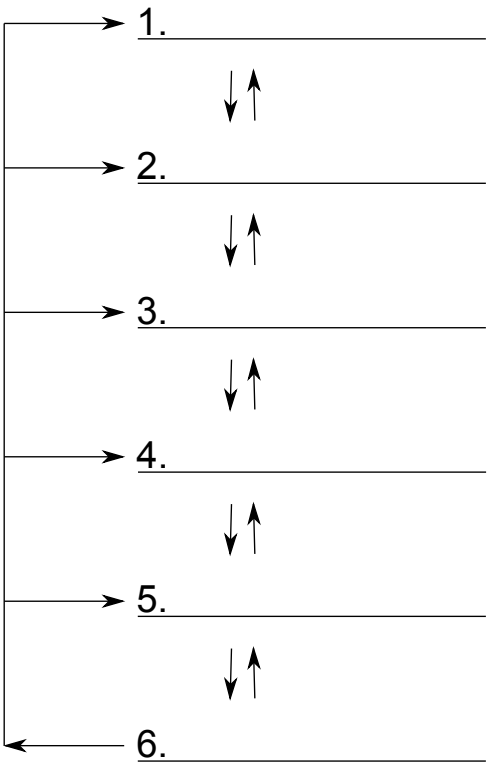
b. Show the math involved! Given what each process knows and the contents of messages, how do they actually synchronize their clocks?

c. What modifications would you have to apply to Cristian's method to ensure perfect time synchronization between the nodes?

Hurry up, the clock on the wall might not be synchronized with yours!

Question 4 (6 points)

Security is an iterative process: you constantly have to be prepared to react to changes that occur in the real world. Perhaps a new attack vector becomes known, or you have to install a new software suite which may require you to update how your organization handles new security-related policies. In this question, we ask you to complete the figure that you should recognize from the lectures on security and briefly state what each step means. You get 0.5 pts for each correct label and 0.5 for each correct explanation!



Question 5 (4 + 4 + 2 points)

Replication can give us increased speed as well as improved fault-tolerance, so it is a Good Thing. However, one size does not fit all, and different replication schemes exists. Choosing the right one for the job is a task for people such as yourself, so let's get to it!

a. Draw a figure and explain how *passive replication* works, and discuss its properties.

b. Draw a figure and explain how *active replication* works, and discuss its properties.

c. The general replication phases are *Request*, *Coordination*, *Execution*, *Agreement*, and *Response*. In the schemes you have just described are there any optimizations you can make in either of these phases for *read-only* operations? If so, state them!

Hint: note that you can get up to 4 whole points for your explanations of how the schemes work. Really show why you deserve them!

Hint 2: we can't really give any better hint than listing the phases in the third point, can we?

This page is intentionally left blank-ish.

Question 6 (3 + 1 + 2 points)

Deadlocks ("I wait for you, while you wait for me — neither of us can continue") are an unfortunate reality for the locking concurrency control scheme unless we cautiously avoid them. If our transactions are distributed, detecting deadlock gets even trickier.

a. Deadlocks can either be detected, or prevented from happening in the first place. A primitive way of prevention is to lock all resources that the transaction will require. Why is this a bad or even (for some systems) impossible way of handling the issue?

b. What is a phantom deadlock?

c. What are vulnerable locks? How do they work?

Write here too!

Question 7 (5 × 1.5 points)

Understanding the terminology used within a subject can sometimes (as within computer security) be fundamental. Not only does the proper use of terminology make someone sound trustworthy and educated, it is also central in understanding new material concerning the subject, and relate this to earlier experiences.

For each pair of security related terms below:

- *briefly* explain *both* terms (0.5 pts per term) and;
- explicitly state if and how they relate to each other (0.5 pts).

The terms are:

a. Cross-site Scripting (XSS) and Peer-to-Peer

b. Key Distribution Center and PKI

c. SSL and Public Key

d. Intrusion Detection System (IDS) and Chinese Wall model

e. Biba security model and Confidentiality

Question 8 (2 + 2 + 2 + 2 points)

Logical time uses "event counters" rather than physical time to order events. Recall that events may be either internal events (i.e. state changes), send events, or receive events.

a. Make a figure with 3 processes that send a few (minimum 5, and each process has to send at least once) messages back and forth and have some internal events. Write the **Lamport logical clock values** for each event. Include some concurrent events!

b. Make a figure with 3 processes that send a few messages (same restriction as above) back and forth and have some internal events. Write the **vector clock values** for each event. Include concurrent events!

c. From a theoretical point of view, values from **Lamport logical clocks** provide us with less reasoning power than values from vector clocks, with regard to which events took place before which in real time. Use your figures to show this!

d. Lamport logical clocks are not without use, though! Show rules for comparing **Lamport logical clocks** that have been extended with totally ordered **process identifiers** that make it possible to get a total ordering of events that occur at the processes! Briefly state how/why this works in your example!

Question 9 (-3 to 3 points)

The following questions require only a true or false answer. Correct answers give 0.5 points, whereas incorrect answers are penalized with -0.5 points. Note that the total from the question may be negative, and this will impact your final score. No answer is the safest option, and counts as 0 points. Any text besides "true" or "false" will not be taken into consideration.

Starvation can not happen if forward validation is used for concurrency control	
A message encrypted using private key X can be decrypted by either the private key X or the public key associated to X	
Lost updates are caused by dirty reads.	
For logical clocks, $e \rightarrow e' \rightarrow L(e) < L(e')$	
Two-phase locking is identical to two-phase commit, except it involves locks	
There are no mutual exclusion algorithms that can survive a single crash failure	