Net ID:

University of Illinois at Urbana-Champaign

Department of Computer Science

Midterm Solutions

CS 425 – Distributed Systems

Fall 2009

Exam duration: 75 minutes

Instructions

Print your name and NetID in the space provided below; **print your NetID in the upper right hand corner of every page.**

Name: _____

NetID: _____

This is a **closed** book, **one page of notes allowed**. You **may** use calculators. Do all parts of all seven problems in this booklet. This booklet should include this title page, plus 7 additional pages. Do your work inside this booklet, using the backs of pages if needed. The problems are of varying degrees of difficulty so please pace yourself carefully and answer the questions in the order which best suits you. The maximum grade on this midterm is **100 points**.

Problem	**Points**	**Score**	**Grader**
1. True & False	15		
2. Time & Clock	20		
3. Ordered Multicast	15		
4. Mutual Exclusion	10		
5. Consensus	15		
6. Failure Detection.	15		
7. P2P	10		
Total	100		

Net ID:

Problem 1 True or False (15 Points)

State whether the following statements are **True** or **False** and provide a 1 sentence justification for your answers in each case.

a) (**3 Points**) Causal ordering implies total ordering and FIFO ordering in an ordered multicast protocol (you don't need to come up with an example, just one sentence justification to argue why the statement is true or false).

False. Causal does not imply total ordering.

b) (**3 Points**) The Chandy-Lamport global snapshot algorithm works correctly for non-FIFO channels.

False. Non-FIFO channels could lead to application messages overtaking the marker, thus leading to the recording of an inconsistent state of the channels.

c) (**3 Points**) In a system with N processes, the Chandy-Lamport snapshot algorithm will always show that at least (N-1) channels are empty.

True. The channels through which each process receives the marker for the first time are recorded to be empty. All such channels define a spanning tree, thus, there are N-1 empty channels.

d) (**3 Points**) It is possible to solve Byzantine Generals problem in an asynchronous system where all the generals (processes) are loyal (correct) but the messages may be dropped.

False. Failed processes are indistinguishable from correct processes with arbitrary message delays. Since asynchronous BG is impossible, so is the above.

e) (**3 Points**) Consider two clocks that drift 1 second in every 106 seconds with respect to each other. A resynchronization interval of 2 x 104 milliseconds is sufficient to limit their skew to 20 milliseconds.

True. Since the clocks resynchronize every 208 ms to skew 0, we have to worry only about an interval of 128 ms at the end of the 106 seconds where a skew could be accumulated 106000/208 = 509 with reminder 128). If the clocks have 1000 ms (1 second) skew after 106 seconds (the worst case as specified), they will have 1.96ms skew after 208 ms. So even in the worst case the skew after 128 ms will be always below 20 ms.

(Note: there are several reasonings that we have accepted as long as they made sense).

Net ID:

Problem 2: Events and Clocks (20 Points)

Let us consider Figure 1.

(a) (2 Points) Show all pairs of concurrent events in Figure 1;

(b) (3 Points) Specify how many possible consistent cuts are there that contain the event **h** .

(c) (5 Points) Is the run <a,b,h,c,d,f,e,g> a linearization of events? Explain why?

(d) (10 Points) Specify Lamport clocks and vector timestamps for each event. You may assume that all logical clocks start initially with all zeros.

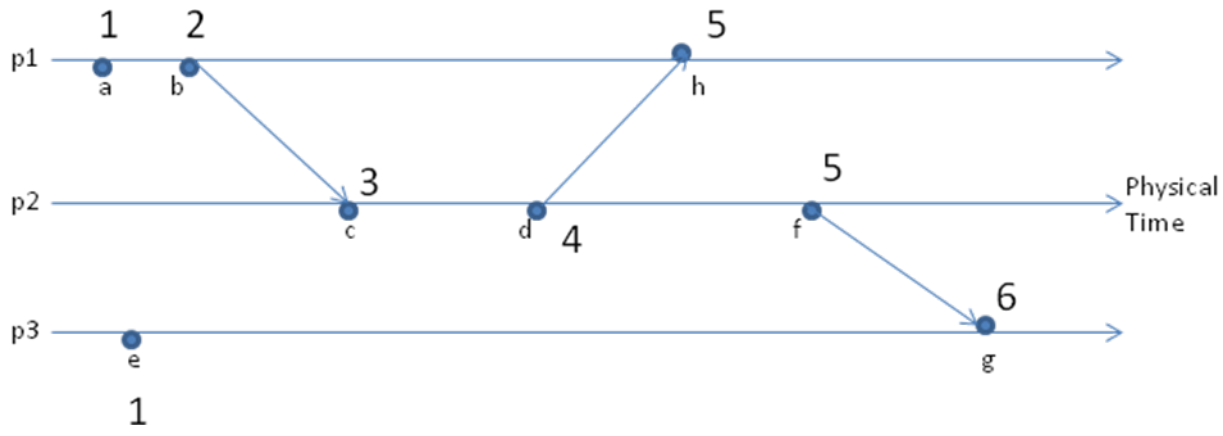
Solutions:

(a) Pairs of concurrent events: <a,e>, <b,e>, <c,e>, <d,e>, <h,e>, <f,e>, <h,f>, <h,g>

(b) Cut 1: a,b,c,d,h; Cut 2: a,b,c,d,h,e, Cut 3: a,b,c,d,h,f; Cut 4: a,b,c,d,h,f,e; Cut 5: a,b,c,d,h,f,g,e

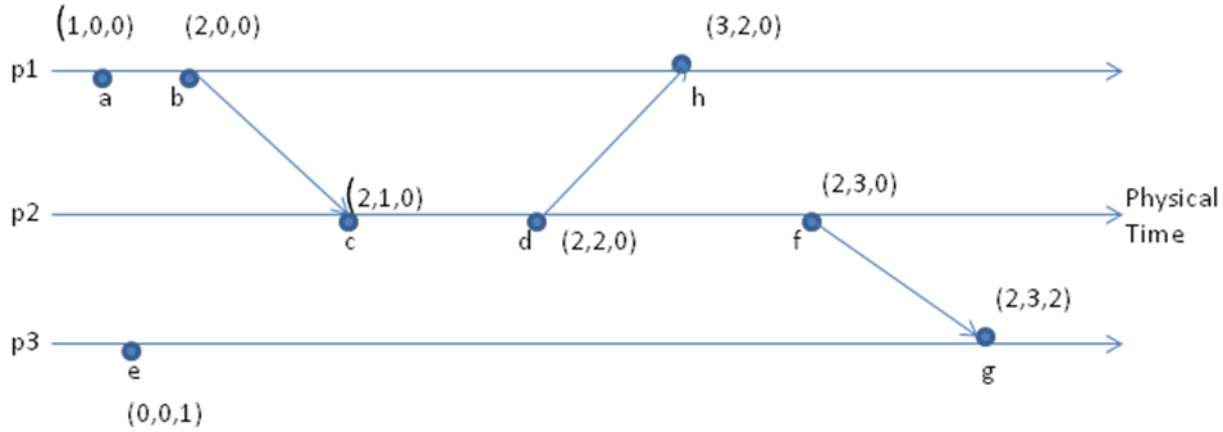
(c) The run is not a linearization of events because **h is specified in the list before event d, but h** did **not happen before d**. For the run to be a linearization of events, all events must be in 'happened-before' relation.

(d) Lamport clocks



(c') Vector Clocks

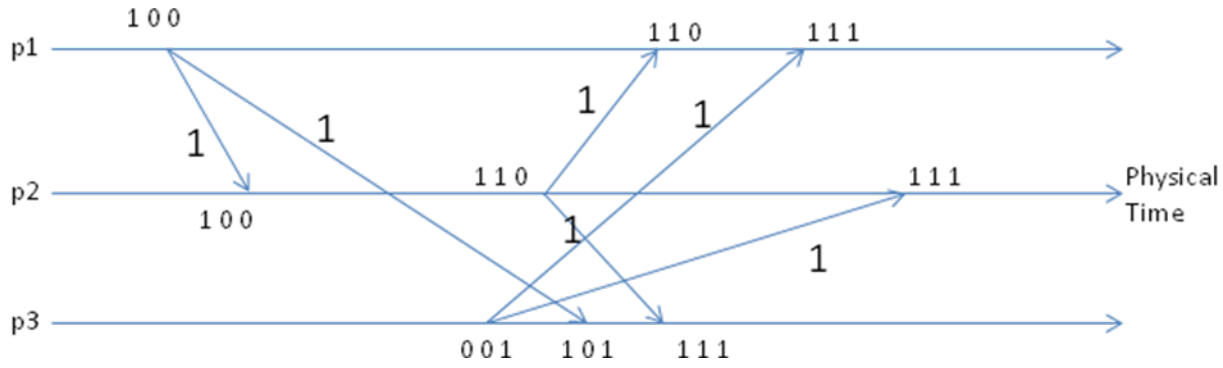
Net ID:



Problem 3: Ordered Multicast (15 Points)

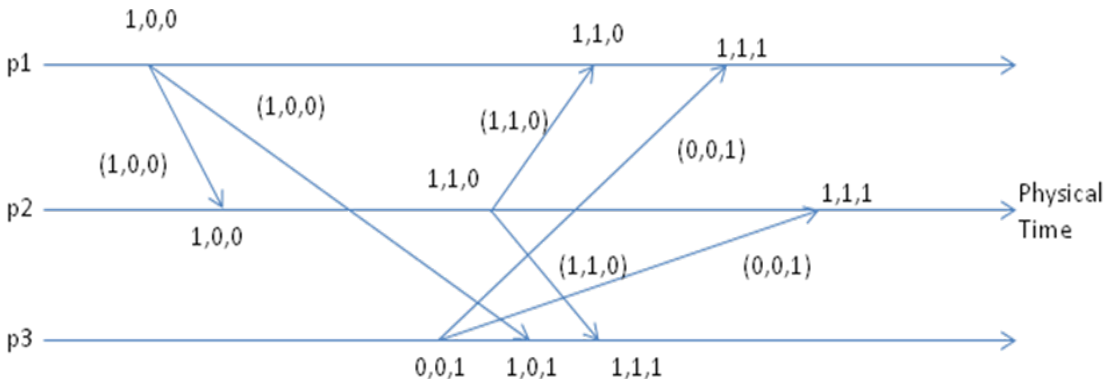
Consider Figure 2. Using sequence numbers (for FIFO ordering multicast) or vector clocks (for Causal Ordering multicast), mark states at the point of each multicast send and multicast receipt. Also mark multicast receipts that are buffered, along with the points at which they are delivered to the application.

(a) FIFO ordered multicast algorithm – all receipts are accepted (none are buffered).



(b) Causal ordering - all accepted – no buffering

Net ID:



Problem 4: Mutual Exclusion (10 Points)

Consider a group of distributed processes, P1, P2, P3, and P4 that share an object. They use the Ricarti-Agrawala algorithm for management of mutual exclusion. P1 is currently in the critical section and there is no other node in the "wanted" state. Now consider requests from P4, P2 and P3 (in that order) to enter the same CS. Note: These requests are also received in this order (P4, P2, P3).

(a) (**5 Points**) Show the state (as required by the algorithm, i.e. "held", "wanted", etc.) and queue entries at each processor.

P1: state: held; Q= {P4, P2, P3}

P2: state: wanted; Q= {P3}

P3: state: wanted; Q = {}

P4: state: wanted; Q = {P2, P3}

(b) (**5 Points**) Now, P1 exits the CS (Critical Section) and informs all relevant nodes that CS is released. Show the state and queue entries at each processor, at this stage.

P1: state: released; Q= {}

P2: state: wanted; Q= {P3}

P3: state: wanted; Q = {}

P4: state: held; Q = {P2, P3}

Net ID:

Problem 5: Consensus (15Points)

Consider **synchronous distributed system**! Is it possible to achieve consensus in spite of crash failures in such a system? If yes, give an algorithm and prove why it works (sketch). If not, give a proof (sketch) to show that it is impossible.

Solution: Answer. Yes, it is possible to solve consensus in a synchronous distributed system with crash failures; the Dolev-Strong algorithm described below achieves this. Suppose the system has N processes, and let f be the maximum number of failures in any execution. The algorithm for process i, i ,1,<,N} is as follows: process i has a variable called Vi, where Vi*k+, k≥0, stores the values known to i in round k; vi is the initially proposed value for i, and di is the decided value. Initially Vi[1] = {vi} Vi[0] = {}; and di = . for each round r (1≤r≤f+1): B - multicast(g, Vi[r] –Vi[r-1]); Vi[r+1] = Vi[r] while (in round r): On B-deliver(Vj) from pj: Vi[r+1] = Vi[r + 1] ∪ Vj after (f + 1) rounds set di= min(Vi[f+1]).

Proof of correctness. Termination is obvious because there are only finite number of rounds and each round is of finite duration. Agreement and integrity is proved by showing that after f+1 rounds for all i,j, Vi[f+1] = Vj[f+1]. Suppose *Vi[f+1] ≠ Vj[f+1].* Then, there is *v Vi[f+1]* such that *v* is not in *Vj[f+1].* This implies that there is a process, say *k*, that delivered *v* to process i in round *f+1* but crashed before delivering *v* to process j. Thus, in the previous round *v Vk[f]* but *v* is not in *Vj[f].* Continuing in the same way, there is a process *l* that delivered *v* to process *k* in *round f* but crashed before delivering *v* to process *j.* And so forth all the way back to *Vj[1].* For the same argument to hold in round 1, we will need f+1 failures, but we have already assumed that there are at most f failures, which is a contradiction. Thus, Vi[f+1] and Vj[f+1] must be identical and the decision values di and dj for correct processes must be the same.

Problem 6: Failure Detection (15 Points)

Consider a variation on the central coordinator algorithm for mutual exclusion. Instead of one coordinator, suppose we have two coordinators, such that the algorithm will continue to operate even if at most one coordinator fails. Suppose that the two coordinators are connected with a channel with maximum delay = D. Other channels are potentially asynchronous.

When a process needs to enter critical section, it sends a request to both the coordinators. The process may enter the critical section when it receives a grant message from either of two coordinators.

Net ID:

What protocol should the coordinators implement to ensure that mutual exclusion and liveness is guaranteed up to a single coordinator failure? If this condition cannot be met, explain why.

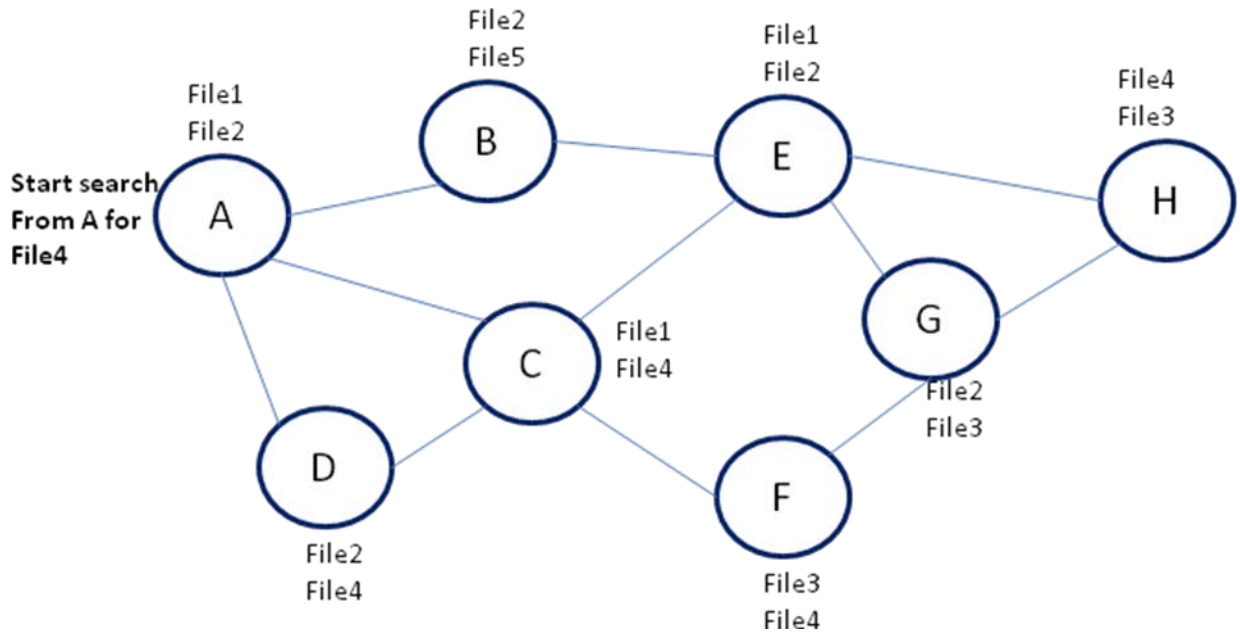
Solution: Let's name the two coordinators **P** and **S,** with **P** being the primary coordinator; and **S**, the secondary coordinator. **S** monitors whether **P** has failed. To achieve this, **P** is required to send a message to **S** every **T** seconds, and if **S** does not receive a message from **P** within **T+D** interval, **S** concludes that **P** is faulty**. P**, before sending a token to another process, informs **S** the identity of the process, say A, to which the token is to be sent. **S** can then record that a request from A has been served. When **S** detects the failure of **P**, the token is either with **P**, or A when **P** failed. By coordinating with **A**, **S** can determine where the token is located, and start serving unserved requests from its queue.

Problem 7: P2P Networks (10 Points)

Consider the Gnutella unstructured peer-to-peer system with the specified files at each peer shown in Figure 3. Each node that has a connection to another node is its neighbor.

(a) (**3 Points**) Specify membership list for each node in the graph of Figure 3.

(b) (**7 Points**) Illustrate in detail how the search algorithm (query/query hit) runs to find **file4**, starting from node A with **TTL=2**. Clearly show the results of the search algorithm at every step and the final result from which node(s) does **A** get **file4**.



Solution:

Net ID:

(a) Membership list on A: (B,C,D), ML on B: (A,E), ML on C: (A,D, E,F), ML on D: (A,C) ML on E: (B, C, G, H), ML on F: (C,G), ML on G: (E, F, H), ML on H: (E, G).

(b) Send query message from A to its neighbor members B,C, D (TTL will be 1 on B,C,D). Nodes C and D will return query hit message since they have **file 4**. Node C forwards query messages to their neighbors E and F (TTL becomes 0). Node D forwards query message to C (this query message gets dropped since C already responded). Node F returns query hit since it has file 4. Node B sends query to E. E drops the query message from B, since it already saw the message query for file 4 from C. Node E does not have any file 4, so it does not return any reply. Also, E will not forward query message since TTL is 0. Query hit message from F returns along the query path F,C,A. Node A will get query hits from nodes C,D and F. A can select the best node with the highest speed.