CSE 486/586 Distributed Systems
Midterm
Monday, 3/5/12

DIRECTIONS

- Time limit: 45 minutes (3:05pm - 3:50pm)
- There are 50 points and 5 bonus points (whatever you score more than 50 will be your bonus points).
- This is a closed-book, no calculator, closed-notes exam.
- The only exception is your cheat sheet (one-sided, letter-sized).
- You should turn in your cheat sheet as well as your answer sheets at the end of the exam.
- Each problem starts on a new page.
- Please use a pen, not a pencil. If you use a pencil, it won't be considered for regrading.
- Each problem also explains its grading criteria. "Atomic" means that you get either all the points or no point, i.e., there are no partial points.

Name:	
UBITName:	

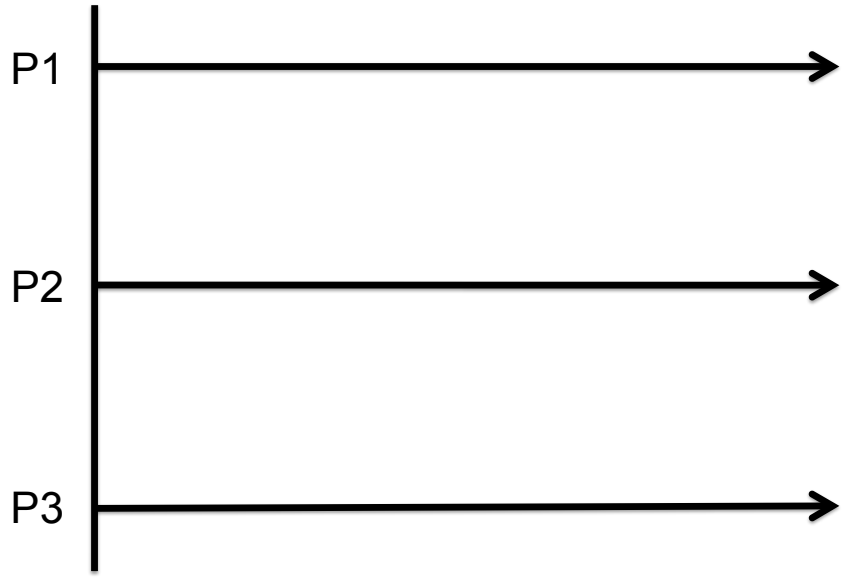
Problem #	Score
1	
2	
3	
4	
5	
6	

1. (a) Explain what fate-sharing is and how it was applied to the design of the Internet.
(**Grading**: atomic 3 points)

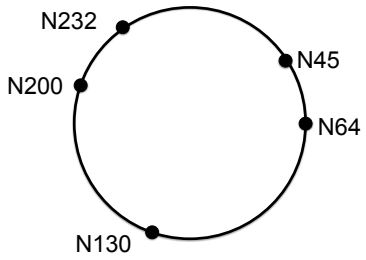
(b) Explain what the end-to-end argument is. Describe this in one sentence and don't forget to include what the exception is.
(**Grading**: atomic 3 points)

(c) Under what conditions can we design a reliable, totally-ordered multicast algorithm?
(**Grading**: atomic 3 points)

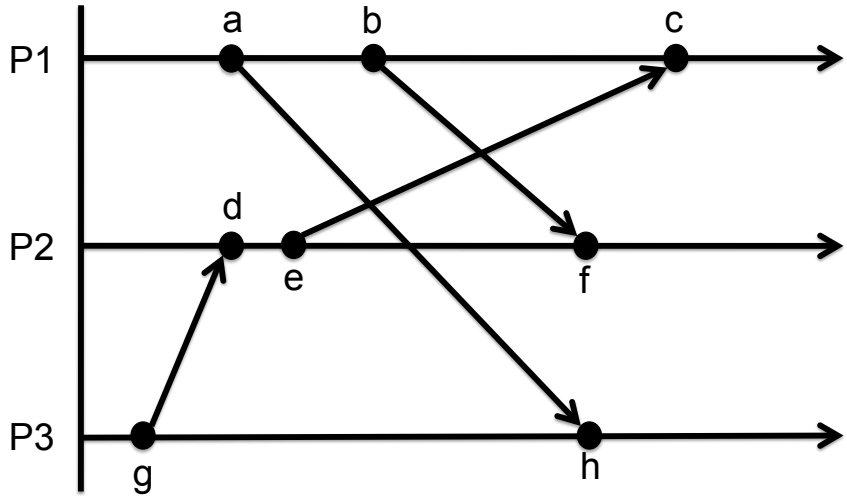
2. In Lamport clocks, it is ***not*** guaranteed that if $L(e) < L(e')$ then $e \rightarrow e'$, where $\rightarrow$ indicates the "happened-before" relation between two events and $L(e)$ denotes the timestamp of event e at whatever process it occurred at. Give an example that demonstrates, even if $L(e) < L(e')$, it is not true that $e \rightarrow e'$, using 3 processes below. Clearly mark which 2 events demonstrate it.
(**Grading:** atomic 4 points)



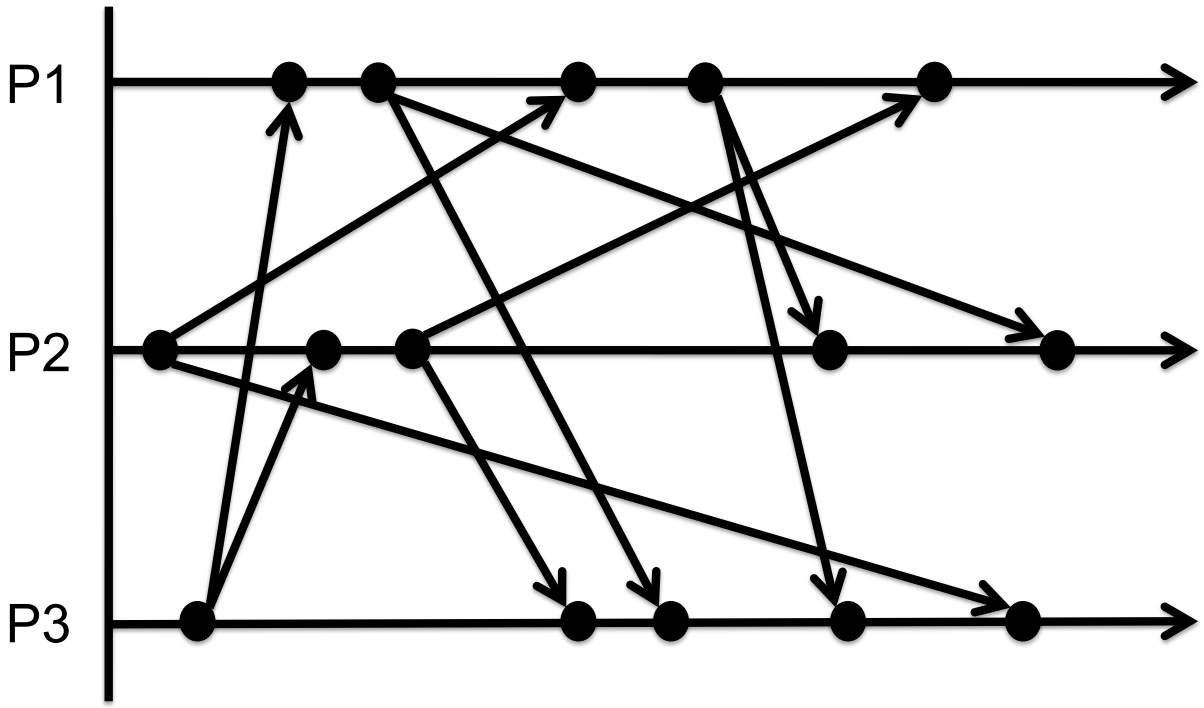
3. Below is a Chord ring with the id space of 2^8. There are 5 nodes in the system. What is the finger table at node 200 (N200)? You can use the table provided below.
(**Grading:** 5 points)



4. Using the vector logical clock discussed in class, list *all possible pairs of concurrent events* that appear in the timeline below. You can use || to denote a pair of concurrent events, e.g., $a \,||\, b$ means a and b are concurrent.
(**Grading:** 12 points)



5. Using the multicast algorithm that provides causal ordering discussed in class, mark the timestamps at the point of each multicast send and each multicast receipt. Also mark multicast receipts that are buffered, along with the points at which they are delivered to the application.
(**Grading:** 10 points)



6. Tired of trying to understand distributed mutex algorithms developed by others, your instructor of CSE 490/590 has decided to write his own algorithm. His algorithm assumes that 1) network channels are reliable and preserve FIFO order; 2) there is no process failure. The following is the algorithm description, assuming that it runs at process p_i. The algorithm uses a local lock which is just meant to be used inside each process; it has nothing to do with distributed mutual exclusion.

#1: On initialization

state := RELEASED;
local_lock := UNLOCKED; ***// Per-process lock***
cnt := 0;
For each process $p_j (j \neq i)$ in the group,
    rcv_j := 0;

#2: To enter the critical section

lock(local_lock);
state := WANTED;
cnt++;
For each process $p_j (j \neq i)$ in the group,
    Send a request to p_j with rcv_j attached to it;
unlock(local_lock);
Wait until (number of replies received = $(N-1)$);
state := HELD;

#3: On receipt of a request from p_j at p_i $(i \neq j)$

lock(local_lock);
tmp_rcv := rcv_i value attached in the request from p_j
if (*state* = HELD)
    Queue the request from p_j without replying;
else if (*state* = WANTED and *tmp_rcv* < *cnt*)
    Queue the request from p_j without replying;
else
    Reply immediately to p_j;
    rcv_j++;
unlock(local_lock);

#4: To exit the critical section

lock(local_lock);
state := RELEASED;
For each queued request,
    // Say the request is from p_j
    Send a reply to p_j;
    rcv_j++;
unlock(local_lock);

(Continue on the next page)

(a) Does this algorithm guarantee liveness? If the answer is yes, prove it. If the answer is no, give a clear example that does not satisfy liveness.
(**Grading:** 5 points)

(b) Does this algorithm guarantee safety? If the answer is yes, prove it. If the answer is no, give a clear example that does not satisfy safety.
(**Grading:** 10 points)