Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.824 Spring 2005

Second Exam

Please write your name on this cover sheet and on any exam pages you detach from this sheet.

Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit.

You have 80 minutes to complete this exam.

THIS IS AN OPEN BOOK, OPEN NOTES EXAM.

Please do not write in the boxes below.

1 (xx/25)	2 (xx/25)	3 (xx/25)	4 (xx/25)	Total (xx/100)
25	23	25	15	88

Name: DAN PORTS

I Part One

10 **1. [10 points]:** Why does Frangipani distinguish between read locks and write locks? Why is that a better approach than having a single kind of lock?

Efficiency, Multiple servers can hold a read lock at one time and hold the data in its cache. If there were only one kind of lock, only one read/write lock could be held at one time, reducing the utility of caching.

15 **2. [15 points]:** Look at the pseudo-code in Section 3.1 of the Ivy paper (Li and Hudak's Memory Coherence in Shared Virtual Memory Systems). Observe this code at the end of the Write Server:

```
IF I am manager THEN BEGIN
  lock(info[p].lock);
  invalidate(p, info[p].copy_set);
```

Suppose that the programmer implementing this code accidentally swaps the lock and invalidate, so that the implementation looked like this:

```
IF I am manager THEN BEGIN
  invalidate(p, info[p].copy_set);
  lock(info[p].lock);
```

Explain a specific situation in which this erroneous code would cause Ivy to produce incorrect memory contents.

In this case, it is possible that during the interval between the invalidate and lock a read request for the same page might arrive and be processed. This can happen because the lock is not held. Then the read-requesting client will have a copy of the page as it existed before modification, but it will not be invalidated. This is incorrect.

25

II Paxos

The Reliable Computing Corporation (RCC) has designed a fault-tolerant airline seat reservation service. Travel agents send reservation requests to the service over the Internet, and the service responds with either a seat reservation or an error indication. Obviously it is vital that the service not assign the same seat to two different passengers.

RCC's system achieves fault-tolerance by replicating the reservation state on three servers. The state consists of a list of the reservation requests that have been granted, in the order that they were granted. Each entry in the list contains the seat number, the passenger name, and a request ID included by the travel agent in each request message. The request ID allows the service to recognize retransmitted requests, and reply with the information from the record that's already in the list, rather than incorrectly allocating a new seat. A server can tell which seats have already been reserved by scanning the reservation list.

RCC uses a replicated state machine to ensure that the servers have identical reservation lists. The three servers use Paxos (see the notes for Lecture 15) to agree on each new reservation to be added to the list. They execute Paxos for each reservation (rather than just using Paxos to agree on a new view and primary after each server failure). Each value that the servers agree on looks like "reservation list entry number 37 assigns seat 32 to passenger Robert Morris with request ID 997." When a server receives a request from a travel agent, it scans its reservation list to find the next available seat and the next list entry number, and uses Paxos to propose the value for that list entry number. The system runs a separate instance of Paxos to agree on the value for each numbered list entry.

The three servers are S1, S2, and S3. S1 receives a request from a travel agent asking for a seat for Aaron Burr. S1 picks Paxos proposal number 101 (this is the n in Lecture 15). At about the same time S2 receives a request asking for a seat for Alexander Hamilton. S2 picks Paxos proposal number 102. Both servers look at their reservation lists and see that the next entry number is 38, and that the next seat available is seat 33. Both servers start executing Paxos as a leader for reservation list entry number 38.

Each of the three sequences below indicates the initial sequence of messages of a possible execution of Paxos when both S1 and S2 are acting as leader. Each message shown is received successfully. The messages are sent and received one by one in the indicated order. No other messages are sent until after the last message shown is received. Your job is to answer these questions about the final outcomes that could result from each of the initial sequences: Is it possible for the servers to agree on Aaron Burr in seat 33 as entry 38? Is it possible for them to agree on Alexander Hamilton in seat 33 as entry 38? For each of these two outcomes, how it could occur or how does Paxos prevent it from occurring?

```
S1 -> S1 PREPARE(101)
  S1 -> S1 RESPONSE(nil, nil)
S1 -> S2 PREPARE(101)
  S2 -> S1 RESPONSE(nil, nil)
S1 -> S3 PREPARE(101)
  S3 -> S1 RESPONSE(nil, nil)
S2 -> S1 PREPARE(102)
S2 -> S2 PREPARE(102)
S2 -> S3 PREPARE(102)
... the rest of the Paxos messages.
```

3 **3. [5 points]:** Answers:

S_1 receives 3 responses, so will choose a value, and send ACCEPTs.

But S_2 has sent PREPAREs with a higher n. So it depends on whether these PREPAREs receive responses containing S_1's chosen value as accepted.

If one of the $S_{1,2,3}$ sends a response with S_1's chosen value to S_2 before S_2 has a majority of replies, ✗ S_2 will send ACCEPTs with S_1's value, and S_1 will be accepted as ~~leader~~ primary. Then Burr gets the seat.

If S_2's PREPAREs are processed by a majority of the nodes, then S_1's accept messages will be rejected because S_2's Proposal number is higher. So S_2 may become the primary, and Hamilton could get the seat.

```
S1 -> S1 PREPARE(101)
  S1 -> S1 RESPONSE(nil, nil)
S1 -> S2 PREPARE(101)
  S2 -> S1 RESPONSE(nil, nil)
S1 -> S3 PREPARE(101)
  S3 -> S1 RESPONSE(nil, nil)
S1 -> S3 ACCEPT(101, ``Aaaron Burr...'')
S2 -> S1 PREPARE(102)
S2 -> S2 PREPARE(102)
S2 -> S3 PREPARE(102)
... the rest of the Paxos messages.
```

5 **4. [5 points]:** Answers:

S_3 has accepted S_1's value since it had not received any other PREPAREs at the time of the ACCEPT. So it will send the accepted value to S_2 in its RESPONSE. If S_2 receives this response before it has a majority of responses, it will ACCEPT and agree upon S_1's value. So Burr gets the seat. ✓

Suppose instead S_3 crashes ~~after~~ before getting the ACCEPT. Then S_2's PREPAREs might arrive at S_1, S_2 before S_1's accepts. Since S_2 has the higher proposal number they will set $n_p = 102$ and hence ignore S_1's ACCEPTs. So S_2 could get a majority to accept, and Hamilton would get the seat.

```
S1 -> S1 PREPARE(101)
  S1 -> S1 RESPONSE(nil, nil)
S1 -> S2 PREPARE(101)
  S2 -> S1 RESPONSE(nil, nil)
S1 -> S3 PREPARE(101)
  S3 -> S1 RESPONSE(nil, nil)
S1 -> S3 ACCEPT(101, ''Aaaron Burr...'')
S1 -> S1 ACCEPT(101, ''Aaaron Burr...'')
S2 -> S1 PREPARE(102)
S2 -> S2 PREPARE(102)
S2 -> S3 PREPARE(102)
... the rest of the Paxos messages.
```

5 **5. [5 points]:** Answers:

S_1 has gotten a majority to accept its value. So S_2 will get responses from these nodes containing S_1's value, and S_2 will use that instead. Burr gets the seat.

It's not possible for S_2 to have its value accepted, since S_1 and S_3 will not send RESPONSEs without S_1's value; if S_2 gets these, it will agree on S_1's value; if it doesn't, then it doesn't have a majority of replies. Hamilton can't get the seat.

10

6. [10 points]: Suppose one of the RCC servers has received an ACCEPT for a particular instance of Paxos (i.e. for a particular reservation list entry), but never received any indication about what the final outcome of the Paxos protocol was. Outline what steps the server should take to figure out whether agreement was reached and to find out the agreed-on value. Explain why your procedure is correct even if there are still active leaders executing this instance of Paxos.

~~This node should contact all the other nodes.~~

The node needs to find out if a majority has accepted a value. It can do this by issuing a proposal to all nodes. If a majority reply with the same accepted value, then this is the agreed-on value. Otherwise agreement hasn't yet been reached. The node can continue the Paxos algorithm in the usual way — sending ACCEPTs if it has a majority of responses.

rather, we don't know if agreement was reached.

This gives the correct result even if there are still active leaders, because it is the same as the ~~2PC~~ multiple-leader case, which we know Paxos handles correctly.

ordTs > valTs on b_2, b_3

III FAB

7. [15 points]: Suppose that a FAB system has three bricks, b_1, b_2, and b_3. There is just one seggroup, and it contains all three bricks. FAB is using replication, as detailed in Figure 4 of the FAB paper. Block x starts out containing value 100 (all replicas have value 100, and all replicas have ordTs equal to valTs). Coordinator c_1 starts to execute write(200) for block x, but crashes after it sends the Write message to b_1 and before it sends Writes to the other two bricks. b_1 receives and processes the Write. This is the only execution of the write() function by any coordinator. Then coordinator c_2 performs two read()s for block x, the second starting after the first completes, and prints the resulting pair of values. For each pair below, indicate whether that pair is possible, and explain how it could arise or how FAB prevents that pair of read values from occurring.

(15)

5 100 100: Yes, & b_1 would have crashed, in which case the first read will initiate recovery and only replies will come from b_2 and b_3, so 100 will be chosen as the new value. The next read will also return 100.

3 100 200: No. When the first read initiates recovery, it c_1 replies, then since it has the highest valTs, 200 will be chosen as the value. So c_1 must not reply. But then if 100 is chosen as the value, it will be written to all bricks with the timestamp of the read, and the second read can't return 200.

3 200 100: No. The value of 200 can only be chosen if c_1 replies during recovery and is written to a majority of bricks. Then 100 can't be read later.

4 200 200: Yes. b_1 can return 200 during the read recovery, and it will be chosen because it has the highest valTs. Then it can be written to the other bricks, and future reads will return 200.

10

8. [10 points]: You're the system administrator of a FAB with three replicated bricks. Your users use the FAB at all times to read and write data. You accidentally step on the Ethernet cable of one of the bricks, which permanently damages the cable and causes the brick to lose its network connection. It will take you about fifteen minutes to run out to CompUSA and buy a replacement Ethernet cable. Before you leave for CompUSA, you can either start a reconfiguration (as in Section 5) or not start a reconfiguration. Which is the best plan? Explain your reasoning. If you would need more information to answer, explain what that information is and how you would use it to decide.

There are two reasons to reconfigure:
1) to prevent NVRAM from being filled because timestamps can't be garbage-collected.
2) to ensure that both nodes are fully synchronized, having all the updates, in case one fails.

The principal reason not to reconfigure is to avoid the overhead of the reconfiguration protocol.

Unless reconfiguration is going to be unusually expensive for some reason, it's best to reconfigure, especially since (2) means that updates could be lost if ~~one~~ another node fails while you're at CompUSA

IV Ruthenian Consistency

You are a member of the Ruthenian People's Popular Front (RPPF), whose aim is to plot the overthrow of the oppressive government of Ruthenia. The RPPF is organized as a large number of "cells," each cell consisting of three members. Each cell knows the members of a few other cells, but no-one knows all the members. However, everyone knows how many cells there are, and each cell knows its own unique ID (a number between 1 and the total number of cells). The total number of cells is constant. The cell structure is intended to make it hard for the government to arrest the whole RPPF, even if they have arrested a few of its members.

From time to time various RPPF cells generate instructions that they want to communicate to every other cell. Cells that know each other can exchange messages written on slips of paper. Thus a message will need to be forwarded between many cells before every cell has a copy.

The RPPF is worried about cells being confused by deceptive orders of arrival of messages. They are particularly anxious to ensure that cells can recognize situations in which two messages are originated simultaneously, by cells that were not aware of the other message. The reason is as follows. Suppose cell c0 receives two messages m1 and m2 that are originated by different cells, and that m1 and m2 contain conflicting instructions. If the originator of m2 knew about m1 before it generated m2, then c0 should ignore m1 and pay attention to m2. But if m1 and m2 were generated without knowledge of each other, then c0 will know that it should exercise extraordinary care when interpreting the two sets of instructions.

More formally, the RPPF wants every cell be able to decide the relative order in which any pair of messages was originated. Suppose cell c0 has a copy of message m1 originated by cell c1, and m2 originated by c2. Cell c0 should be able to decide whether c2 had a copy of m1 when it originated m2, or c1 had a copy of m2 when it originated m1, or neither had a copy of the other message. Call these situations "m2 is after m1," "m1 is after m2," and "m1 and m2 are concurrent."

Your job is to design an ordering system for the RPPF.

Each cell is allowed to keep a copy of each message it sees, and can also maintain any state it can derive from those messages. The only form of communication allowed to a cell is passing written notes to the cells it knows about. Your solution should continue to operate even if the Ruthenian government eliminates a few cells, so it should not depend on any special cells. You can assume that the government is not aware of your communication system.

You should try to minimize the amount of communication required by your system; it would not be good if each message had to include a huge amount of book-keeping information.

15

9. [25 points]: Please explain how your ordering system works. You should describe the contents of each message, the state that each cell must keep, any communication protocol the cells must adhere to, and the algorithm a cell should use to decide whether one messages is after another or whether two messages are concurrent.

Each cell should number the messages it sends with a strictly increasing counter. i.e. $m_{C_i,1}$ is the first message from cell C_i.

Each cell should keep a version vector of the highest-numbered message it has received from each cell, and include a copy of its version vector in each message.

Then messages can be compared by checking their timestamps. If $m_{C_a,x}$ was generated before $m_{C_b,y}$, then the bth elt of the vector timestamp will be at least y.

If the vector Ts's aren't comparable, the messages are concurrent.

End of Exam