Name: **AndrewID:**

15-440 Homework #1 (Spring 2012)
Due: Tuesday, February 28, 2012

Communication

1. Consider a sliding window protocol, such as the one we discussed in class. What effect does the window size have upon throughput? Please derive a formula that expresses the maximum *throughput* (bits/second) of the connection between two systems as a function of the *bit rate* of the channel, the one-way *latency* of the channel (seconds), the *window size* (number of frames), and the *frame size* (bits).

2. Again consider the window size of a sliding window protocol. What factor(s) influence the appropriate window size? What happens if the window is too small? Too large?

3. Java's RMI considers all parameters, except remote object references to be input parameters. They are passed by copy and not returned. In the context of the Java environment, why is this a necessity? In other words, why can't Java emulate a pass-by-reference as a pass by in-out, as was done in C with RPC?

4. In Java objects are very straight-forward to decompose and reconstruct. .class files provide blue-prints for each type of object, and Java has rich functionality for extracting the properties of an object, including all of its state.

   As a result, it would seem that the serialization of an object for use as a parameter or return value would require nothing from the object, itself. But, it doesn't work this way. Java's RMI can only send *Serializable* objects as parameters or return values. These classes of objects implement their own serialization methods.

   Assume that .class files could be sent to a client, as needed, via HTTP, as is done with stubs. Why doesn't the Java RMI just parse the objects, flatten the fields, and serialize the objects automatically? In your discussion, you should include at least one example of a problematic type of object.

Time

5. Assume that the real-time clocks within some population of *N* workstations can drift, at most, 10 seconds per day. Consider the task of synchronizing these clocks, to within 0.01 seconds of each other, using each of *Cristian's Algorithm* and *The Berkeley Algorithm*.

   How many messages are required for each approach?

6. In class we discussed using a host id, such as IP address, to break ties and impose a total ordering on Lamport time stamps. This approach, on its face, is unfair. Some hosts are more likely to win than others. We could alleviate this unfairness by randomly choosing between two concurrent events.

   a) Consider the voting districts algorithm. What problem could arise if two nodes ordered two concurrent events differently? Explain how this could occur.
   b) How could you ensure that every node in the system agrees on this randomly chosen ordering of concurrent events? (i.e. what would the clocks need to do)
   c) What are the advantages of giving up fairness?
   d) Which approach do you like better? Why?

7. In class we discussed vector timestamps as a tool for detecting causality violations, but we didn't discuss certain trade-offs present in the design of the approach. The algorithm that we discussed in class sends the full vector of timestamps with each message. This can be wasteful if the sending processor has received very few messages since the last time it sent a message to the same host. In this case, most (if not all) of the entries remain unchanged.

   A protocol can be designed that reduces the size of the vector timestamp by sending fewer of these uninteresting entries, but this savings comes at the expense of overhead, including storage and processing, on each host.

   Please assume that the hosts have plenty of processing power and O(P) space available to store state information related to the timestamp protocol. Operating under these assumptions, design an algorithm for compressed vector timestamps that reduces the size of the vector timestamps as much as possible, without affecting the utility of the time-stamping. Please describe the resulting approach.

Mutual Exclusion

8. Fix the code below so that the numbers print in order, using simple mutexes or "counting" semaphores:

```
#include <pthread.h>
#include <stdio.h>

void* thread_one (void* args){
  printf("1");
  printf("5");
  return NULL;
}

void* thread_two (void* args){
  printf("2");
  printf("4");
  return NULL;
}

int main(){
  pthread_t tid_one;
  pthread_t tid_two;

  pthread_create(&tid_one, NULL, thread_one, NULL);
  pthread_create(&tid_two, NULL, thread_two, NULL);

  printf("3");

  pthread_join(tid_one, NULL);
  pthread_join(tid_two, NULL);

  printf("6\n");
}
```

9. In class, we discussed several techniques for enforcing mutual exclusion. But, we didn't discuss techniques for enforcing other concurrency control policies. Please adapt the majority vote scheme to solve the "at most n users" mutual exclusion problem. In other words, please adapt it so that it allows not more than "n" users to access the shared resource at a time. Your new design should be as loyal to the original design as is practical.

   If it does not make sense to use an adaptation of the voting scheme for this purpose, please explain the reason.

10. In class, we discussed several techniques for enforcing mutual exclusion. But, we didn't discuss techniques for enforcing other concurrency control policies.

    Please select one among the mutual exclusion protocols from our in class discussion and adapt it to enforce reader-writer locks, instead of mutual exclusion. Needless to say, you should select a protocol which can be usefully adapted for this purpose. Your new protocol should be as loyal to the original as is practical.

    A read-writer lock allows readers to share an object with other readers, but ensures that a writer excludes both other writers and also readers. Sometimes this protocol is described as "Shared reads, exclusive writes". Readers should starve writers.

11. What are the relative costs and benefits of mutual exclusion by voting and mutual exclusion by token ring?

12. What do you believe is the “Best” distributed mutual exclusion algorithm, balancing the message-passing cost and robustness. Why? What are its relative costs and benefits as compared to a central approach?

Replication

13. Consider the quorum-based protocols that we discussed in class. Each of these protocols requires some type of read-write locking to ensure that the client is playing with the same version of the object at each of the participating replicas. Without this, two writes could interleave or a read and a write could interleave.

    Please describe an efficient locking protocol to address these concerns. Assume that the client can interact with all servers, except in the event of failure.

14. Consider a system with *N* replicas, a read quorum of *R*, and a write quorum of *W*. How much failure can this system endure? Please express your answer in terms of *N, R,* and *W*.

15. Consider the quorum-based replication algorithms we discussed in class. What quorum policy or policies guarantee that all versions read will be the same and the most up-to date? Why?

16. Consider the use of physical timestamps and version numbers for consistency control in replication. What are the costs and benefits of each technique. Please specifically the bounds that each technique is able to place on the freshness of the object received by any one client, the consistency of the objects as simultaneously viewed by multiple clients, and the message-passing costs of each technique.