15-440 Homework #2 (Spring 2010)

Due: Thursday, March 4th, 11:59 pm

Turnin Instructions: submit your homework to

```
/afs/andrew/course/15/440-sp10/handin/hw2/username/
```

You must create the folder with your username. If you mess up and need to submit again, create another folder, bobjones.1 or bobjones.2. Kesden's script will kill everything but the latest one and remember the dot between the username and the number. Also submit in PDF format!

---

1) 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 frequently should the clocks be synchronized for each approach?

2) In class we discussed *Lamport Logical Time* and a vector time protocol based on the same event counting scheme. Do these two approaches necessarily order events in the same way?

   If so, please explain why this must be the case. If not, please explain why not and provide a simple figure to illustrate one such case.

3) 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.

4) 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.

5) The Java RMI system implements a blocking paradigm for remote method invocation. In other words, the caller is blocked until the remote method executes and returns the result. Consider a non-blocking version of the RMI. Specifically, assume that after a remote method is invoked, the flow of control in the caller continues, and that the calling thread is informed about the availability of the result via an exception. The exception handler (catch block) then takes appropriate action.

   Please describe the impact of the cost and benefits of this approach to the RMI, both from the perspective of the RMI implementor and the application developer. You may neglect the case where the return type is void.

6) 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?

7) 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.

8) AFS and Coda make use of a stateful server and callbacks to protect consistency. If these mechanisms are sufficient to ensure global consistency, please describe how they enforce this invariant. If not, please explain the unprotected window of vulnerability, in other words, explain how a consistency problem can arise despite a functioning server properly issuing callbacks

9) Some distributed file systems use a combination of a distributed hash and replication to scatter multiple copies of data blocks across the storage nodes. Please explain how the replication component enhances both the availability and throughput of these systems.

10) Why are many modern distributed file systems implemented completely as user-level processes, rather than by integration with the kernel, as by a kernel module or virtual file system?

11) How does eliminating, or deprioritizing in-place edits of files, as compared to appends, simplify the design of a distributed file system (in such cases where the workload allows this)?