# 1. Disconnected Operation

The Coda system provides support for disconnected operation, in which clients are able to continue accessing data during temporary disconnection from (or failure of) the file servers.

a. There are two basic families of approaches for controlling access to replicas in a distributed system that can suffer from network partitions: *optimisitic* and *pessimistic*. Briefly describe how both optimistic and pessimistic replica control work and the advantages and disadvantages of each approach.
b. The Coda system uses optimistic replica control so that clients can access files when disconnected. What data is *likely* to be in the client's cache when it is first disconnected? Which of this data is *guaranteed* to be in the cache and why can this differ?
c. While disconnected, what types of errors can occur on a Coda client and why? When reconnected, what types of errors can occur and why?

# 2. Authentication and Authorization

Kerberos is an authentication mechanism that provides the client name and a shared session key to the server. Kerberos, as it stands, does not allow users to change names.

a. Why doesn't Kerberos support name changes?

Suppose you wanted to add support for users (but not servers) to change their name. This means that the client can log in with their new name, but servers will still give them the appropriate access to resources.

b. What has to change about the way clients acquire tickets?
c. What has to change about the way servers use the client name passed in tickets?

# 3. File System Interfaces

As disks have become larger, many operating systems have incorporated indexing and query interfaces to enable users to find their files faster. These systems typically index files (including file contents and metadata) offline, so recently created files may not be found. A possible future design for file systems is to do away with conventional file system naming (hierarchies of directories and file names) and adopt a database approach, where all file access is done through queries over a flat set of files using indices. Each file could be annotated with attributes providing additional information, such as what collections it is a member of, what program and user created it, date and time information, or documents that it links to.

a. Are there important properties of the current file naming mechanism that a query mechanism cannot provide (or has difficulty providing)? What are they?

Indexing files can take place synchronously, when a file is modified, or can be deferred until the system is idle.

b. When does indexing files synchronously make sense (i.e., for what kinds of workloads)?
c. Why might you want to defer indexing files?

# 4. Microkernels and Reliability

Microkernels have been around since the earliest days of computing systems (e.g., Brinch-Hansen's Nucleus, 1969). A microkernel based operating system divides its major components (e.g., file system, memory manager, scheduler) into distinct processes, using messaging primitives to communicate between these components.

a. One of the major advantages claimed by proponents of microkernels is higher reliability, due to the protection provided between major OS subsystems. What hardware features does a microkernel based system use in order to provide protection between processes?
b. How do these protections differ from what typical operating systems do to provide protection between processes?
c. Given these protections, what aspects of reliability does a microkernel improve?
d. Given these protections, what aspects of reliability does a microkernel make worse?

# 5. Synchronization

Java provides a form of monitor-based synchronization with condition variables. These monitors provide a "synchronized" key word for class functions (methods) that provides a mutual exclusion lock with all other synchronized methods. Each object (class instance) is a separate monitor, so there is a monitor lock for each object.

The interesting twist is that a Java monitor instance can have only one condition variable.

a. Provide a monitor-based solution to the bounded-buffer (producer/consumer) problem using monitors that allows only a single condition variable in each monitor. You can use Java syntax or C++-like syntax with monitors added.
b. The one-condition restriction to Java monitors can lead to inefficient programs with excessive numbers of process switches. Why do such problems arise? How can you avoid these problems by restructuring the code into a solution with multiple monitors?
c. Discuss whether the one-condition-per monitor restriction in Java is a minor inconvenience or a serious deficiency.

# 6. Network File Systems

Consider the NFS (version 2) and AFS network file systems.

a. NFS has remote-mount semantics to access a file from a client computer and AFS provides access to a file to each client computer from a global name space.

   What are the advantages and disadvantages of each of these two naming schemes?

b. An NFS server needs to keep little state about what each client computer is doing with each open file, and AFS keeps detailed information about each open file.

   What are the advantages and disadvantages of each of these two state-tracking schemes?

c. Updates to an open file caused by writes to that file are handled differently by NFS and AFS.

   Describe these differences.

   What are the advantages and disadvantages of each of these two update schemes?