Problem 1

How many lookup calls are necessary to resolve a five-part pathname (for example /usr/users/sam/../abc.txt) for a file that is stored on an NFS server? What is the reason for performing the translation step-by-step?

Answer: 5 lookup calls. The reasons is that the name may cross a mount point at the client, the directories holding different parts of a multi-part name may reside in file systems at different servers.

Problem 2

Why should UFIDs be unique across all possible file systems? How is uniqueness for UFIDs ensured?

Answer: Because file groups may be moved and distributed systems that are initially separate can be merged to form a single system. UFIDs can be made unique by including the 32-bit IP address of the host that creates them and a 16-bit integer derived from the date, producing a unique 48-bit integer.

Problem 3

Explain in which respects DSM is suitable or unsuitable for client-server systems.

Answer: DSM is unsuitable for client-server systems in that it is not conducive to heterogeneous working. Furthermore, for security we would need a shared region per client, which would be expensive. DSM may be suitable for client-server systems in some application domains, e.g. where a set of clients share server responses.

Problem 4

Discuss whether message passing or DSM is preferable for fault-tolerant applications.

Answer: Consider two processes executing at failure-independent computers. In a message passing system, if one process has a bug that leads it to send spurious messages, the other may protect itself to a certain extent by validating the messages it receives. If a process fails part-way through a multi-message operation, then transactional techniques can be used to ensure that data are left in a consistent state. Now consider that the processes share memory, whether it is physically shared memory or page-based DSM. Then one of them may adversely affect the other if it fails, because now one process may update a shared variable without the knowledge of the other. For example, it could incorrectly update shared variables due to a bug. It could fail after starting but not completing an update to several variables. If processes use middleware-based DSM, then it may have some protection against aberrant processes. For example, processes using the Linda programming primitives must explicitly request items (tuples) from the shared memory. They can validate these, just as a process may validate messages

Problem 5

Discuss whether the following operations are idempotent :

1. Pressing an elevator call button
2. Sorting a list
3. Appending to a file

Is it a necessary condition for idempotence that the operation should not be associated with any state?

Answer:

Problem 5 continued on next page...

1. Idempotent (elevator stays in requested state)
2. Idempotent (sorting a sorted list does not change the sorted list)
3. Non-idempotent (two appends will add twice as much data)

It is not necessary that the operation is not associated with state, as the sort example shows.

Problem 6

Describe a distributed computation that can be implemented using MapReduce. (Please come up with something on your own, as opposed to an example you have seen elsewhere). Sketch the pseudocode for the map() and reduce() functions.

Answer:

Problem 7

Provide an example of a distributed computation that would be difficult to implement in MapReduce, giving full reasons for your answer.

Answer: