DSM: Tutorial 8

Question 1

- Manageability: user-oriented names will allow administrators to more easily interpret and remember the components they are dealing with.
- Openness: user-oriented names can help new developers to become familiar with the system, but machine-oriented names can reduce conflict and potential confusion.
- Fault tolerance: user-oriented can help errors to be better understood.
- Performance: machine-oriented names can be shorter and avoid the lookup cost associated with user-oriented alias names.
- Scalability: machine-oriented names have a much larger range of potential values.
- Security: on the one hand, user-oriented names can improve security as a side-effect of improving manageability, but machine-oriented names can make the system more opaque and harder to attack (security through obscurity - not usually an actual defence).
- Transparency: user-oriented names can improve transparency in the case of aliases that allow the implementation to be moved and/or changed without the user knowing, but one could also argue that in a transparent system the user should not need to know the names of things, so machine-oriented names should be allowable.
- Concurrency: there are no arguments for either side.

Question 2

- The `open` request creates a file handle. If the client or server crashed before this handle was closed, one or both host states will become inconsistent (the server will forget the client's cursor position, or the client will leave the handle open forever). The same applies for the read lock that will be placed on the file.
- The name `sample.txt` is not absolute and relies on some previously specified context (although such does not appear in the code).
- Two calls are made for each loop, each of which will introduce latency. It would be better if the **read** operation returned some data and a flag to indicate whether or not there was any more data to be read.
- One byte is read per loop, which is very inefficient because the client will block and unblock for each byte, and many times more data will be spend on the overhead of requesting and sending that single byte.

Question 3

Middleware could provide location transparency without providing migration transparency if the migration affected characteristics that the client needed, other than location (such as transfer protocol or authentication system).

Middleware could provide migration transparency without providing location transparency if the migration didn't affect components of location that the client is required to be aware of.

Question 4

In a very large distributed system, starting up a host might mean mounting many external directories, which can be slow and expensive. This mounts must also be maintained, which adds an overhead.

This can be mitigated by using lazy auto-mounting. The available mounts are specified, but external systems are not actually mounted until the client requests a file from within them. Another approach could be to use a virtual file system that minimises mounting and handles retrieval of remote files via the network, where appropriate.