Project 1 RPC Design Document

Serialization Protocol

On the client side, each package is a contiguous memory block of three components in the order listed below:

- The size of the package (size_t)
- An unique identifier of the RPC agreed upon between the client and the server (int)
- All the function arguments passed in by values

The size of the package is sent first so that the server is aware of how many bytes it should be expecting. This is to handle the short counts of TCP connection due to its internal buffering and delays. The server side will then read the next integer to decide on which function to invoke locally before deserializing the corresponding function arguments and performing the local call.

After the local function call, the server will send back the return code to the client first. Based on the return code, the client side can determine whether the RPC is successful. If the RPC fails, the client will receive and set errno accordingly. Otherwise, the client will, if necessary, receive an additional package as specified by the return code and set other parameters according to the function definition (e.g. the actual bytes read for the read RPC).

Recursive Serialization and Deserialization of dirtreenode

This protocol follows the implementation introduced during the recitation session. Two traversals are needed in my implementation. The first traversal checks the size in bytes of the buffer to hold the serialized tree. The tree will be serialized in pre-order and each serialized node contains three contiguous memory blocks in order:

- the length of the name string (int)
- the number of children nodes it has (int)
- the actual name string

The server side will send back the serialized tree and free the memory used upon success. The client needs to recursively deserialize each tree node accordingly following pre-order traversal order.

Handling Concurrency

The server will fork() a child process to handle all the RPC calls of a client's lifetime after a connection is accepted over the socket. The parent process will close the connection and start waiting for new connections from other clients. Before the client program exits, it will send a special message to the child process so that it can terminate and close the connection socket.

Distinguishing Between Local & Remote File Descriptors

A large offset of 100000 is used to differentiate a local file descriptor (i.e. one created not by the open RPC call such as a connection socket) and a remote file descriptor created on the server. More specifically, this large offset will be added to all the remote file descriptors on the server side before sending them back to the client. In this way, the client side function call can decide on whether to invoke a local call or a remote call by comparing the file descriptor with the offset.