/***
Transparent Remote File Operations
***/

This library allows the remote file operation procedure calls on a server connected to a client via TCP. TCP was used in order to ensure reliability.

Serialization Protocol:
****

For any function call, the parameters are serialized in the following general format

||OPCODE|| PARAMETERLEN|| _____PARAMETERS_____||

||_____ANY VARIABLE LENGTH PARAMETERS_____||

Response from the Server:

|| RETRUN VALUE || ERRNO||

||___ANY VARIABLE LENGTH PARAMETERS__||

The decision for choosing such message format and a few other design decisions are explained below.

Design Decisions on the client side:
****

1. One connection per client. That is, when the client calls any library function from mylib.c for the first time, a TCP connection is established with the server. This connection is used for all subsequent communications. This was done in order to reduce connection overhead for each and every library call. The connection is closed via the _fini function.

2. Instead of sending all the parameters in one shot, a few of the parameters (such as the write buffer contents) are delivered in a second parameter message. Since the content is of variable length, the server needs to know how many bytes it is to expect from the client. Hence, the almost every other parameter, along with the write buffer content length are sent to the server at first, followed by the actual buffer contents.

3. Nagle's algorithm was disabled using setsockopt call in order to send TCP messages with a small length. This was done in order to reduce the latency.

4. #pragma pack(0) was used in order to take away the structure alignment rules. However, it was found that if any type such as size_t and off_t was used, the alignment rules were not relaxed. Hence in order to fill in the bytes used for alignment purposes in such cases, a filler variable was used. (It doesn't occupy any extra space) however, it takes away the uninitialized buffer send problem.

Design Decisions on the server side:
****

1. Nagle's algorithm was disabled for the same purpose mentioned above.

2. A process per client basis. This form of concurrency was used in order to avoid thread related synchronization overheads. Every child process gets an identical copy of the file descriptors of the parent. Hence, it becomes easy to just transfer the connection handling from the parent to child.

3. Each child is killed when the client tries to close the connection.

4. In some routines, the return values are of variable length. Hence two or more return messages are used in such cases in order to let the client know beforehand on what to expect.

5. For the getdirtree method, the tree is traversed in a pre-order fashion and the node values are sent. This makes it easier than transferring all the nodes on to a single buffer and then dealing with unpacking the tree on the client side. This made it very simple and easy to implement.

6. The message format of every function (as mentioned on the header comment of each client function) was chosen such that the functions on the server only have to deal with the required function parameters. The data such as OPCODE and parameterlength are not passed on the callee function inside the server and are dealt with only inside the main function of the server.