Andrew Koulogeorge
15440 Lab1 Design Document

Serialization Protocol

While each of the interposed functions contains different arguments and thus requires a different byte sequence to be sent over the network, the overall structure I used to send packets between the client and the server is the same.

For each packet, I first calculate the total number of bytes required in the packet based on the size of each argument to be sent. I begin each packet with the total size of the packet as a null terminated string followed by the payload which includes an opcode to decipher which procedure to call on the server and the arguments required for function call.

There are two edge cases to note. The first is read which has a variable number of parameters. I handle this by serializing the number of arguments and adding it directly into the packet along with the function arguments. The second is that some functions have variable length arguments such as the name of files. To correctly identify the length of these variable length inputs, I take the difference between the size of the packet encoded in the header and the number of bytes taken up by the fixed length arguments to the function. This method works for this interposition library since no function that needs to be implemented has more then one variable length argument.

For sending packets back to the client, a similar structure is used as when sending packets to the server: length of entire payload, method's return value and the errno are included in every return packet. Followed this is procedure call specific data (ex: bytes read from a file). To encode the dirtree structure into a packet, I performed preorder traversal on the tree and dynamically copy each node's attributes into a byte sequence. Upon the packet's arrival back to the client, I performed another preorder traversal to allocate space for each node and write the corresponding data back into the treenode structure.

Design Choices

Instead of writing the length of the packet in the header as a null terminated string, I could have instead stored the length as an 8 byte size_t. This method requires slightly less space for large packet sizes and also is also marginally simpler to encode and decode.
I handled potential failures in memory allocation with a custom implementation of malloc called xMalloc that ensures pointers returned are valid. I handle concurrency on the server side by forking off a child to handle the request of each user working with the interposition library. Each child is contained in its own while loop that can handle several RPC calls from the client in its life span. This child worker terminates when the client process breaks the connection and is afterwards reaped by the parent. Upon inspection in valgrind, my encoding of the dirtree structure was not compatible with the libraries freedirtree method so I implemented custom functionality in freedirtree to free all allocated memory in the structure.