15440/15640 project 1 documentation

Author: Qiyang He andrewID: qiyangh

0. General:
    a. Server will receive an integer-length data (which is the operation code) first to decide what kind of request it is, then call corresponding function to handle request.
    b. Client and server will always have the same bytes of data being sent/received in order to keep system safe and stable.

1. Function open:
    a. Client side: Pack opcode(integer), flags(integer), mode(mode_t), length of path(integer), path (char array) into a buffer, send to server; receive fd(integer) and errno(integer) from server, add offset onto fd to indicating an rpc-based file descriptor.
    b. Server side: Deserialize params by their specific size, dynamically set buffer for path name, add a null-character to path string; pack fd, errno, send to client.
2. Function read:
    a. Client side: Pack opcode(integer), fd(integer), number of bytes to read(integer) into a buffer, send to server; receive length of content read(size_t), errno(integer), content (char array) from server, use loop to make sure receive content up to total bytes.
    b. Server side: Deserialize params by their specific size, dynamically set buffer for read content, use loop to send back content to make sure every byte has been sent.
3. Function close:
    a. Client side: Pack opcode(integer), fd(integer) into a buffer, send to server; return value(integer), errno(integer) from server.
    b. Server side: Deserialize params by their specific size, send back return value and errno value to client.
4. Function write:
    a. Client side: Pack opcode(integer), fd(integer), number of bytes to write(size_t) and content (char array) into a buffer, send to server. Use loop to send every byte of content; receive length of content wrote(integer), errno(int) from server.
    b. Server side: Deserialize params by their specific size, dynamically set buffer for write content, send back length of content being wrote and errno value.
5. Function lseek:
    a. Client side: Pack opcode(integer), offset(off_t), whence(integer) into a buffer, send to server; receive new offset(off_t), errno(integer) from server.
    b. Server side: Deserialize params by their specific size, send back new offset and errno.
6. Function stat:
    a. Client side: Pack opcode(integer), ver(integer), path length(integer) and path(char array) into a buffer, send to server; receive return value(integer) and updated stat struct(if success) from server.

b. Server side: Deserialize params by their specific size, dynamically set buffer for pathname, send back return value and stat struct (if success).

7. Function unlink:
    a. Client side: Pack opcode(integer), length of pathname(integer), pathname (char array) into a buffer, send to server; receive return value and errno value from server.
    b. Server side: Deserialize params by their specific size, dynamically set buffer for pathname, pack return value, errno value and send back.
8. Function getdirentries:
    a. Client side: Pack opcode(integer), fd(integer), number of bytes to read(integer), basep pointer(off_t) into a buffer, send to server; receive directory content length, and then receive content by specified length, update basep pointer.
    b. Server side: Deserialize params by their specific size, set buffer based on content length and other return value length, pack content length, errno value, updated basep pointer and content into buffer, send back to client.
9. Function getdirtree:
    a. Client side: Pack opcode(integer), path length(integer), path(char array) into a buffer, send to server; receive serialized tree data, build tree recursively by dynamically allocate tree structure, child array structure and pointers and assign data to each tree node from serialized response of server, return tree root node.
    b. Server side: receive path name, build tree by calling origin function. Serialize tree by sorting out valid data(name, name length, child number) when looping through the whole structure; send back tree size and serialized tree data to client(if success).
10. Function freedirtree:
    a. This function is not an rpc call, because after server send back response for function getdirtree, it has to free the whole tree structure before next rpc request. The freedirtree function actually will only be useful for local calls.