Packet structure for all the operations:

1. read
client->server: | total size | operation type | fd | size |
server->client: | bytes read | errno |

2. write
client->server: | total size | operation type | fd | size | buf |
server->client: | size written | errno |

3. open
client->server: | total size | operation type | flags | m | path length | path |
server->client: | fd | errno |

4. close
client->server: | total size | operation type | fd |
server->client: | return value | errno |

5. unlink
client->server: | total size | operation type | pathname length | pathname |
server->client: | return value | errno |

6. __xstat
client->server: | total size | operation type | ver | path length | path |
server->client: | return value | errno | stat_buf |

7. lseek
client->server: | total size | operation type | fd | offset | whence |
server->client: | return value | errno |

8. getdirentries
client->server: | total size | operation type | fd | nbytes | basep |
server->client: | return value | errno | buf | basep |

9. getditree
client->server: | total size | operation type | path length | path |
server->client: (recursively creating the tree) | name length | name | num_subdir |

Other design decisions:

- I wrote my own freedirtree function so that I can free the memory allocated for the tree completely (and pass the MEM test). My freedirtree only frees the tree in the client side.
- I disabled the Nagle's algorithm because sometimes I need to use several send in one request.