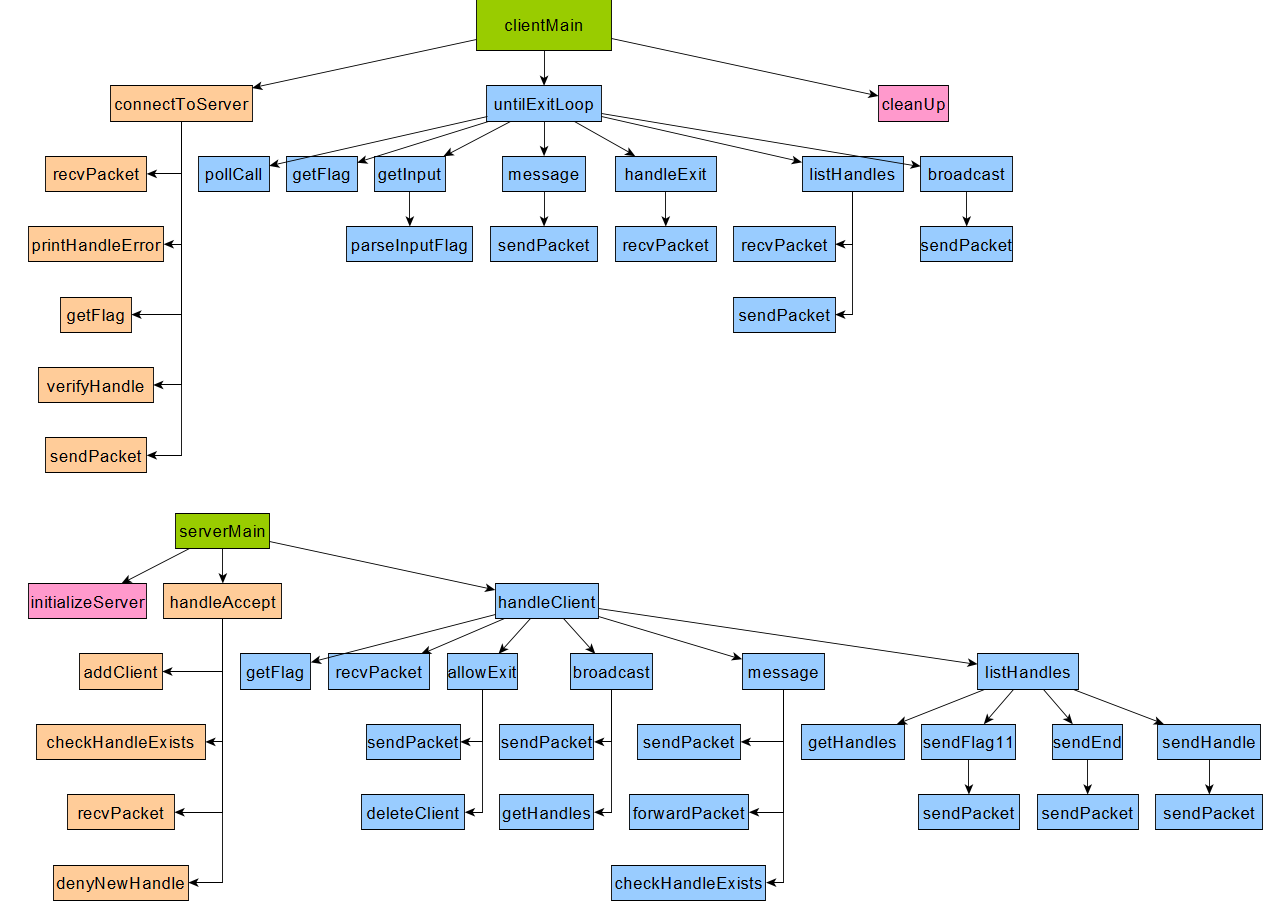
1. **Handle Table data structure on the server.**

|  |  |  |
| --- | --- | --- |
| **Socket/Handle State on the Server** | **How can this happen (what can cause this on the server)** | **If you looked at your socket/handle table, how do you know the socket/handle is currently in this state (what in your data structure would tell you this)** |
| Client socket is opened but handle not valid | A client attempted to connect with a bad handle, but the server did not close the socket. | I would indicate this with a null value for the handle in the table. |
| Client Socket open and client handle is valid | A client successfully connected to the server with a valid handle | Entry in the socket/handle table for the open client and valid handle. |
| Socket was opened and had a handle, but then the socket was closed | Client successfully connected then exited. | No entry for the handle or socket |

* getSocket(handle) -> returns socket number associated with that handle or null if there is no such handle
* getHandle(socket) -> returns the handle associated with that socket if it exists, else null
* addSocketHandle(socketNumber, handle) -> returns 0 if the handle is invalid, 1 on success: checks for valid handle then creates a new entry in the table for socketNumber/handle and adds the socketNumber to the poll set
* closeSocketHandle(socketNumber) -> returns 0 if no such socket exists, 1 on success: closes socket, removes socket from poll set, and deletes entry in table
* getAllHandles(char \*\*list) -> returns number of handles and fills given list with the handles

## Part II – Hierarchical drawing and listing common code

1. **Hierarchical drawing:** 
2. **Common Code**:

* Flag macros
* Uint16\_t getPacketLen(char \* buf)
* Int getFlag(char \* buf)
* sendPacket()
* recvPacket()
* pollLibrary