**Aim**

The primary aim of this program is to establish a **remote command and control channel** between a client machine and a server machine. Specifically, the **client** (the machine to be controlled, usually the one *behind a firewall*) initiates a connection to the listening **server** (the controller), allowing the server to execute arbitrary shell commands on the client machine remotely.

**Algorithm**

**Server (server.py)**

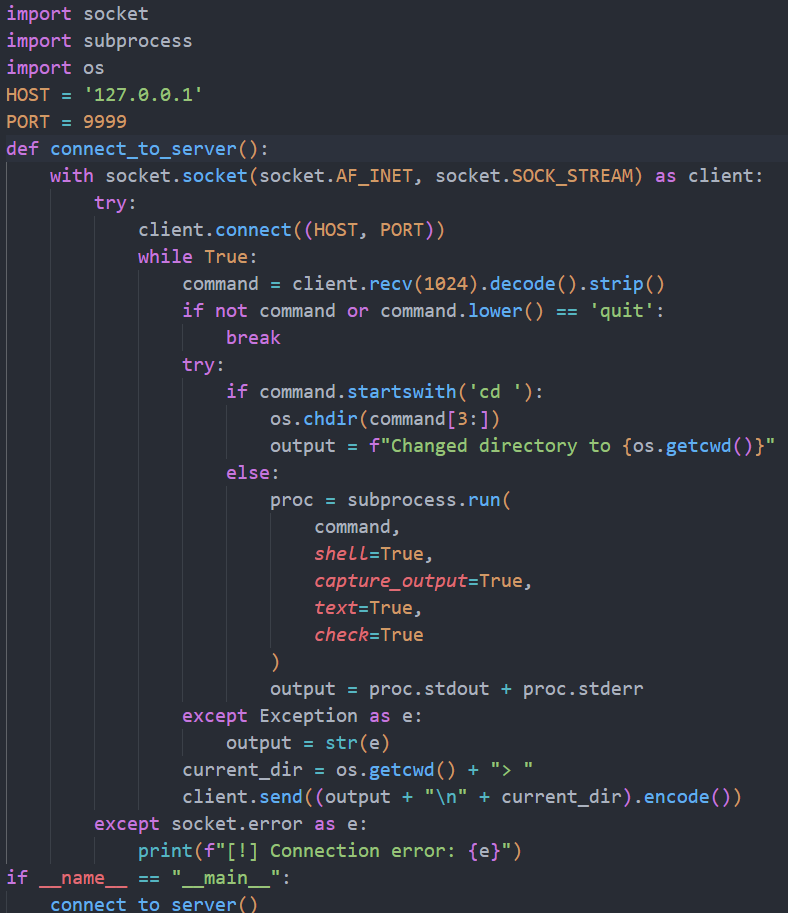
1. **Initialize:** Create a TCP socket and bind it to a specified host and port.
2. **Listen:** Start listening for incoming connections.
3. **Accept:** When a connection arrives, accept it, establish a dedicated thread (handle\_client) for that connection, and wait for more connections.
4. **Interact:** Inside handle\_client:
   * Prompt the user for a command.
   * **Send** the command over the socket to the client.
   * **Receive** the command output (and the client's current directory prompt) from the client.
   * **Print** the received output.
   * Repeat until the command is 'quit'.

**Client (client.py)**

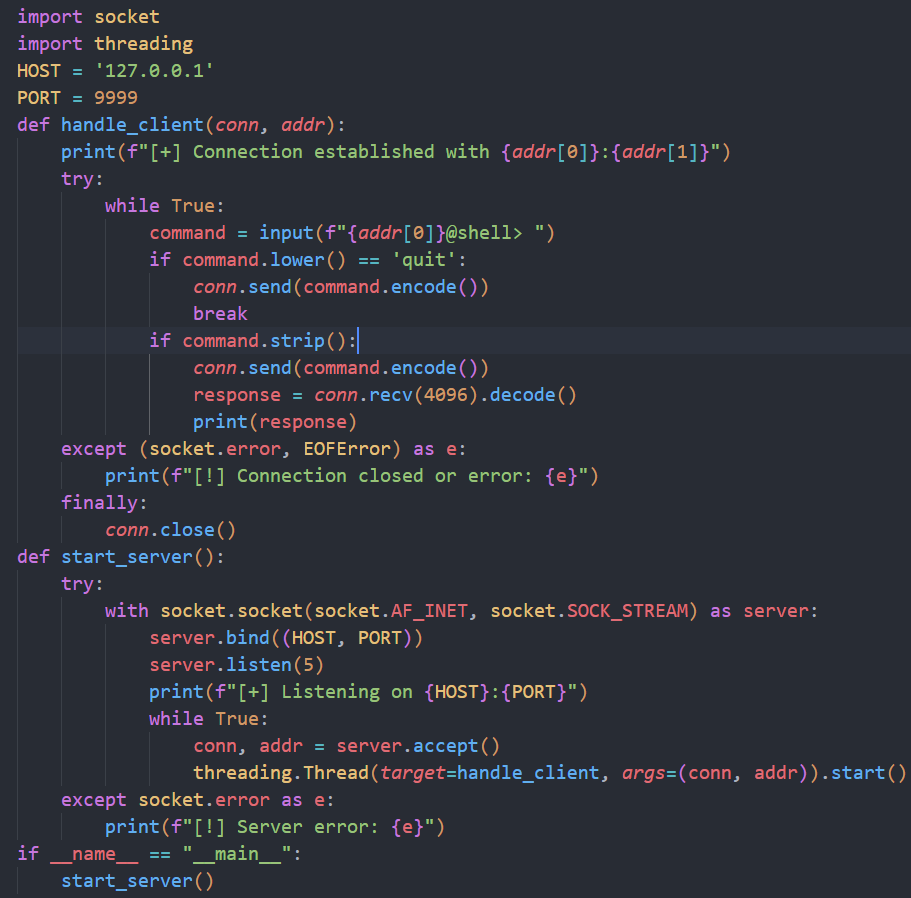
1. **Initialize:** Create a TCP socket.
2. **Connect:** Initiate a connection to the server's specified host and port (this is the "reverse" part).
3. **Receive:** Start a loop to continuously **receive** a command from the server.
4. **Execute:**
   * If the command is 'quit', break the loop.
   * If the command starts with 'cd ', change the current working directory (os.chdir).
   * Otherwise, execute the command using a subprocess (subprocess.Popen), capturing both standard output and standard error.
5. **Send:** Combine the command's output with the client's current directory prompt and **send** it back to the server.
6. Repeat until the connection is closed.

**Source Code**

**Client.py**



**Server.py**

****

**Result**

The successful execution of this program results in a **fully interactive shell session** where the **server** user can remotely execute operating system commands on the **client** machine.

