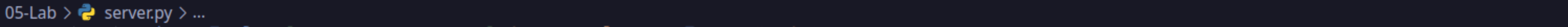


```
[abhishek@hp in repo: CN/05-Lab on ? main [!+] via ? v3.11.3 took 4ms]
python server.py
[SERVER] Keep the files to send/recieve inside folder: 'server/'
[STARTING] Server is listening on :53535
[NEW CONNECTION] 127.0.0.1:54832 connected.
[ACTIVE CONNECTIONS] 1
[127.0.0.1:54832] Create file 'file.txt'
[127.0.0.1:54832] Contents of 'file.txt': hello world from client :)
(ip:port)> 127.0.0.1:54832
(file_name)> file_s.txt
[127.0.0.1:54832] File name recieved
[127.0.0.1:54832] File contents of 'file_s.txt' recieved
[DISCONNECTED] 127.0.0.1:54832 disconnected.
[ACTIVE CONNECTIONS] 0
(ip:port)> 
```

```
[abhishek@hp in repo: CN/05-Lab on ? main [!+] via ? v3.11.3 took 4ms]
python client.py
[CLIENT] Keep the files to send/recieve inside folder: 'client/'
[CONNECTED] Client connected to 127.0.0.1:53535
(file_name)> file.txt
[SERVER] File name recieved
[SERVER] File contents of 'file.txt' recieved
[SERVER] Create file 'file_s.txt'
[SERVER] Contents of 'file_s.txt': hello world from server :)
(file_name)> QUIT!

[CLIENT] Client is shutting down...

[abhishek@hp in repo: CN/05-Lab on ? main [!+] via ? v3.11.3 took 40s]
,
```



```
1  import socket
2  import threading
3  import os
4  import time
5  import readline
6  from urllib.parse import quote, unquote # for encoding/decoding data in url format
7  from dataclasses import dataclass
8
9  # IP = socket.gethostname(socket.gethostname())
10 IP = ''
11 PORT = 53535
12 ADDR = (IP, PORT)
13 SIZE = 1024
14 FORMAT = "utf-8"
15 DISCONNECT_MESSAGE = "QUIT!"
16
17 # MSG Format:
18 #   MSG/<msg>
19 #   CREATE/<file_name>
20 #   POST/<file_name>/<file_content>
21 #   GET/<file_name>
22
23 folder_path = "server/" # folder to store files for server
24
25 clients = [] # list of all clients connected to server
26
27 # Client class to store client connection and address
28 @dataclass
29 class Client:
30     conn: socket.socket
31     addr: str
32
33
34 # find client by address
35 def find_client(addr):
36     for client in clients:
37         if client.addr == addr:
38             return client
39     return None
40
```

```

41
42 current_prompt = "" # store current prompt of input
43
44 def print_msg(msg):
45     if not current_prompt:
46         print(msg)
47     else:
48         input_buffer = readline.get_line_buffer() # store current input buffer
49         print(f"\r{msg}\n{current_prompt}{input_buffer}", end="", flush=True) # print message and restore input buffer
50
51
52 def input_msg(string):
53     global current_prompt
54
55     # set current prompt and take input
56     current_prompt = string
57     result = input(f"\r{string}")
58     current_prompt = ""
59
60     return result
61
62
63 # send message to client
64 def send_msg(conn, msg):
65     conn.send(f"MSG/{quote(msg)}.encode(FORMAT))
66
67
68 # take input from server
69 def server_input():
70     while True:
71         to_addr = input_msg("(ip:port)> ").strip()
72         if to_addr == DISCONNECT_MESSAGE: # disconnect server
73             end_server()
74         elif to_addr in ["", "list", "ls"]: # list all clients
75             print_msg("Active clients:")
76             for client in clients:
77                 print_msg(f" {client.addr}")
78             continue
79
80         to_client = find_client(to_addr) # find client by address

```

```

81     if to_client is None:
82         print_msg(f"Error: Client '{to_addr}' not found.")
83         continue
84
85     msg = input_msg("(file_name)> ") # take file name
86     if msg == DISCONNECT_MESSAGE:
87         end_server()
88     elif msg == "":
89         continue
90
91     # send file name to client
92     to_client.conn.send(f"CREATE/{quote(msg)}.encode(FORMAT))
93
94
95 # handle the file requests from client
96 def handle_file_request(client, request, body):
97     # CREATE/<file_name>
98     if request == "CREATE":
99         file_name = unquote(body) # decode file name
100         file_path = folder_path + file_name
101         print_msg(f"[{client.addr}] Create file '{file_name}'")
102         with open(file_path, "w"): # create file
103             msg = f"File name recieved"
104             send_msg(client.conn, msg) # send acknowledgement to client
105             time.sleep(0.1)
106             client.conn.send(f"GET/{quote(file_name)}.encode(FORMAT)) # send GET request to client to get file contents
107
108     # POST/<file_name>/<file_content>
109     elif request == "POST":
110         file_name, file_content = map(unquote, body.split("/", 1)) # decode file name and content
111         file_path = folder_path + file_name
112         print_msg(f"[{client.addr}] Contents of '{file_name}': {file_content}")
113         with open(file_path, "a") as f:
114             f.write(file_content) # write file content to file
115             msg = f"File contents of '{file_name}' recieved"
116             send_msg(client.conn, msg) # send acknowledgement to client
117
118     # GET/<file_name>
119     elif request == "GET":
120         file_name = unquote(body) # decode file name

```

```

121     file_path = folder_path + file_name
122     try:
123         with open(file_path, "r") as f: # read file content
124             file_content = f.read()
125             client.conn.send(f"POST/{quote(file_name)}/{quote(file_content)}.encode(FORMAT)) # send file content to client
126     except Exception as e: # if file not found
127         err = f"Error: {e}"
128         print_msg(err)
129         send_msg(client.conn, err)
130
131     else:
132         send_msg(client.conn, f"Error: Invalid request '{request}'")
133
134
135 # handle client connection
136 def handle_client(conn, addr):
137     print_msg(f"[NEW CONNECTION] {addr} connected.")
138
139     while True:
140         msg = conn.recv(SIZE).decode(FORMAT) # receive message from client
141         if not msg or msg == DISCONNECT_MESSAGE:
142             break
143
144         try:
145             request, body = msg.split("/", 1) # split request and body
146         except Exception as e:
147             err = "Error: Invalid request format."
148             print_msg(f"[{addr}] {err}")
149             send_msg(conn, err)
150             continue
151
152         if request == "MSG": # if message request
153             print_msg(f"[{addr}] {unquote(body)}")
154             continue
155
156         try: # handle file request
157             handle_file_request(find_client(addr), request, body)
158         except Exception as e: # if any error while handling file request
159             err = f"Error: {e}"
160             print_msg(f"[{addr}] {err}")

```



```

161         send_msg(conn, err)
162         continue
163
164     conn.close() # close connection
165     clients.remove(Client(conn, addr)) # remove client from list
166     print_msg(f"[DISCONNECTED] {addr} disconnected.")
167     print_msg(f"[ACTIVE CONNECTIONS] {threading.active_count()-3}")
168
169 def main():
170     server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # create server socket
171     server.bind(ADDR) # bind server socket to address
172     server.listen() # start listening for connections
173     print_msg(f"[STARTING] Server is listening on {IP}:{PORT}")
174
175     server_input_thread = threading.Thread(target=server_input)
176     server_input_thread.start() # start server input thread
177
178     while True:
179         conn, addr = server.accept() # accept connection
180         addr = f"{addr[0]}:{addr[1]}"
181
182         clients.append(Client(conn, addr)) # add client to list
183
184         thread = threading.Thread(target=handle_client, args=(conn, addr))
185         thread.start() # start client thread
186
187         print_msg(f"[ACTIVE CONNECTIONS] {threading.active_count()-2}")
188
189
190 def end_server():
191     print_msg("\n[EXITING] Server is shutting down...")
192     for client in clients: # send disconnect message to all clients
193         client.conn.send(DISCONNECT_MESSAGE.encode(FORMAT))
194     os._exit(0)
195
196 if __name__ == "__main__":
197     if not os.path.exists(folder_path): # create folder if not exists
198         os.makedirs(folder_path)
199     print_msg(f"[SERVER] Keep the files to send/recieve inside folder: '{folder_path}'")
200     try:

```

```
201     main()
202 except KeyboardInterrupt: # if server is stopped
203     end_server()
```





```

1  import socket
2  import threading
3  import os
4  import time
5  import readline
6  from urllib.parse import quote, unquote # for encoding/decoding data in url format
7
8  IP = socket.gethostbyname(socket.gethostname())
9  # IP = "172.16.19.141"
10 # IP = ''
11 PORT = 53535
12 # PORT = 8006
13 ADDR = (IP, PORT)
14 SIZE = 1024
15 FORMAT = "utf-8"
16 DISCONNECT_MESSAGE = "QUIT!"
17
18 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # create client socket
19 connected = True # flag to check if client is connected to server
20
21 # MSG Format:
22 #   MSG/<msg>
23 #   CREATE/<file_name>
24 #   POST/<file_name>/<file_content>
25 #   GET/<file_name>
26
27 folder_path = "client/" # folder to store files for client
28
29 current_prompt = "" # store current prompt of input
30
31 def print_msg(msg):
32     if not current_prompt:
33         print(msg)
34     else:
35         input_buffer = readline.get_line_buffer() # store current input buffer
36         print(f"\r{msg}\n{current_prompt}{input_buffer}", end="", flush=True) # print message and restore input buffer
37
38
39 def input_msg(string):
40     global current_prompt

```

```

41     # set current prompt and take input
42     current_prompt = string
43     result = input(f"\r{string}")
44     current_prompt = ""
45
46
47     return result
48
49
50 # send message to server
51 def send_msg(conn, msg):
52     conn.send(f"MSG/{quote(msg)}.encode(FORMAT))
53
54
55 # handle file requests from server
56 def handle_file_request(request, body):
57     # CREATE/<file_name>
58     if request == "CREATE":
59         file_name = unquote(body) # decode file name
60         file_path = folder_path + file_name
61         print_msg(f"[SERVER] Create file '{file_name}'")
62         with open(file_path, "w"): # create file
63             msg = f"File name recieved"
64             send_msg(client, msg) # send acknowledgement to server
65             time.sleep(0.1)
66             client.send(f"GET/{quote(file_name)}.encode(FORMAT)) # send GET request to server to get file contents
67
68     # POST/<file_name>/<file_content>
69     elif request == "POST":
70         file_name, file_content = map(unquote, body.split("/", 1)) # decode file name and file content
71         file_path = folder_path + file_name
72         print_msg(f"[SERVER] Contents of '{file_name}': {file_content}")
73         with open(file_path, "a") as f:
74             f.write(file_content) # write file content to file
75             msg = f"File contents of '{file_name}' recieved"
76             send_msg(client, msg) # send acknowledgement to server
77
78     # GET/<file_name>
79     elif request == "GET":
80         file_name = unquote(body) # decode file name

```

```

81     file_path = folder_path + file_name
82     try:
83         with open(file_path, "r") as f: # read file contents
84             file_content = f.read()
85             client.send(f"POST/{quote(file_name)}/{quote(file_content)}.encode(FORMAT)) # send file contents to server
86     except Exception as e: # if file not found
87         err = f"Error: {e}"
88         print_msg(err)
89         send_msg(client, err)
90
91     else:
92         send_msg(client, f"Error: Invalid request '{request}'")
93
94
95 def handle_server():
96     global connected
97     while connected:
98         msg = client.recv(SIZE).decode(FORMAT) # recieve message from server
99         if not msg or msg == DISCONNECT_MESSAGE:
100             connected = False
101             end_client()
102
103         try:
104             request, body = msg.split("/", 1) # split request and body
105         except Exception as e: # if invalid request format
106             err = "Error: Invalid request format."
107             print_msg(err)
108             send_msg(client, err)
109             continue
110
111         if request == "MSG": # if message request
112             print_msg(f"[SERVER] {unquote(body)}")
113             continue
114
115         try: # handle file request
116             handle_file_request(request, body)
117         except Exception as e: # if error occured while handling file request
118             err = f"Error: {e}"
119             print_msg(err)
120             send_msg(client, err)

```

```

121         continue
122
123     client.close() # close connection
124     print_msg(f"[DISCONNECTED] Server disconnected from Client.")
125     os._exit(0)
126
127 def main():
128     client.connect(ADDR) # connect to server
129
130     print_msg(f"[CONNECTED] Client connected to {IP}:{PORT}")
131
132     server_thread = threading.Thread(target=handle_server)
133     server_thread.start() # start thread to handle server
134
135     global connected
136     while connected:
137         msg = input_msg("(file_name)> ").strip() # take input from user
138         if msg == DISCONNECT_MESSAGE:
139             end_client()
140
141         client.send(f"CREATE/{quote(msg)}.encode(FORMAT)) # send CREATE request to server
142
143     client.close() # close connection
144     print_msg(f"[DISCONNECTED] Client disconnected from {IP}:{PORT}")
145
146
147 def end_client():
148     client.close() # close connection
149     print_msg("\n[CLIENT] Client is shutting down...")
150     os._exit(0)
151
152 if __name__ == "__main__":
153     if not os.path.exists(folder_path): # create folder if not exists
154         os.makedirs(folder_path)
155     print_msg(f"[CLIENT] Keep the files to send/recieve inside folder: '{folder_path}'")
156     try:
157         main()
158     except KeyboardInterrupt: # if client is stopped
159         end_client()

```