Experiment-9

Ronanki Trivikram

GITAM

Abstract

This project implements a simple client-server architecture using Python's socket programming. The server listens on port 5005 and responds to client requests for text files. Upon receiving a filename, the server checks for its existence and sends the file contents back to the client. If the file is not found, it returns an appropriate message. The client connects to the server, requests a file, displays the received content, and saves it to a new file prefixed with "received_". This implementation demonstrates basic networking concepts and file handling in Python.

Server code

```
1 usage
def start_server(host='localhost', port=5005):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")

    while True:
        conn, addr = server_socket.accept()
        print(f"Connection from {addr}")
        try:
            # Receive the filename from the client
            filename = conn.recv(1024).decode()
            print(f"Requested file: {filename}")

            if os.path.isfile(filename):
                with open(filename, 'r') as file:
                    content = file.read()
                conn.sendall(content.encode())
            else:
                conn.sendall(b"Sorry, file not found.")
        finally:
            conn.close()

if __name__ == "__main__":
    start_server()
```

Client code

```python
import socket
1 usage
def request_file(filename, host='localhost', port=5005):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))

    # Send the filename to the server
    client_socket.sendall(filename.encode())

    # Receive the response from the server
    response = client_socket.recv(4096).decode()

    # Display the response
    print("Received from server:")
    print(response)

    # Save the content to a new file
    with open(f"received_{filename}", 'w') as file:
        file.write(response)

    client_socket.close()


if __name__ == "__main__":
    filename = input("Enter the name of the file to request: ")
    request_file(filename)
```

Output

Server code

```
"C:\Users\AD Survey\Desktop\python\va
Server listening on localhost:5005
```

Clint code

```
Enter the name of the file to request: IPS.txt
Received from server:

Anup Kumar Singh 30 September 2020

M A Ganapathy 29 February 2024

Nalin Prabhat 15 August 2024

B. Srinivasan Incumbent
```

Explanation

Server Code Explanation

1. **Imports**: The code imports the **socket** module for network communication and the **os** module to check file existence.
2. **Server Initialization**: The **start_server** function initializes a TCP/IP socket, binds it to **localhost** on port **5005**, and starts listening for incoming connections.
3. **Connection Loop**: The server enters an infinite loop, accepting connections from clients. Each connection is handled in a separate block.
4. **Client Connection**: Upon accepting a connection, the server retrieves a new socket for communication (**conn**) and the client's address (**addr**).
5. **Filename Reception**: The server receives the requested filename from the client, decoding it from bytes to a string.
6. **File Existence Check**: The server checks if the requested file exists using **os.path.isfile()**.
7. **File Handling**:
    - If the file exists, it opens the file, reads its contents, and sends the content back to the client after encoding it to bytes.
    - If the file does not exist, it sends a "Sorry, file not found." message.

8. **Connection Closure**: After handling the request, the server closes the connection to the client.

Client Code Explanation

9. **Client Initialization**: The **request_file** function initializes a TCP/IP socket and connects to the server at **localhost** on port **5005**.
10. **Filename Transmission**: The client sends the requested filename to the server, encoding it to bytes.
11. **Response Reception**: The client waits to receive the server's response, reading up to 4096 bytes and decoding it to a string.
12. **Response Display**: The client prints the response received from the server, which could be the file contents or an error message.
13. **File Saving**: The client saves the received content to a new file, prefixed with "received_", ensuring the original file is not overwritten.
14. **Connection Closure**: After completing the operations, the client closes the socket connection.
15. **Demonstration of Concepts**: This implementation showcases basic networking concepts, file handling, and client-server communication in Python, providing a practical example of socket programming.

These points summarize the key functionalities and flow of the server and client code, highlighting how they interact to achieve the desired file transfer.

Thank you.