**Internet and Higher Layer Protocol**

**“CS656-102”**

**Project Report On**

**Remote Desktop Application Using**

**TCP/IP**

**BY:**

**SAI SUMANTH GOUD PARAKALA sp2927  
MANIDEEP JAGADHABI mj457**

**RUSHIL RAO KOULAMMAGARI rk862**

**CONTENTS**

1. **Introduction**
   * Overview of the remote desktop application
   * Objectives and goals of the project
2. **Analysis**
   * Architecture and Approach
3. **Code**
4. **Output Screenshots**
   * Proof of working project
5. **Conclusions**
   * Key takeaways from the remote desktop application
   * Scope for further improvement

**Introduction**

The remote desktop application is a powerful tool that allows users to access and control a remote computer from anywhere in the world, as long as there is an internet connection. The application is designed to be simple and easy to use, with a command-line interface that enables users to execute a variety of remote desktop functions.

Using the remote desktop application, users can connect to a remote computer and access its files. The application supports a variety of operating systems, including Windows, Linux, and macOS.

One of the key features of the remote desktop application is its ability to provide a secure connection between the user and the remote computer. The application uses strong encryption to protect data transmitted over the network and provides authentication mechanisms to ensure that only authorized users can access the remote computer.

In addition, the remote desktop application is designed to be highly reliable and efficient, with a low-latency connection that provides a smooth and responsive remote desktop experience. Whether you're working from home or traveling on the go, the remote desktop application makes it easy to stay connected to your work and your team, no matter where you are.

**Top of Form**

**ANALYSIS**

The Remote Desktop application is designed using a client-server architecture that ensures reliable and secure communication between the client and server. The application provides a user-friendly interface that allows users to connect to their remote computers with ease and perform various tasks, such as file creation, deletion, and transfer.

The server is designed to handle one connection at a time, ensuring that all communication is reliable and efficient. The use of socket programming and encryption ensures that all data is transmitted securely between the client and server. Additionally, the authentication feature ensures that only authorized users can access their remote computers, preventing unauthorized access.

The file transfer feature is also implemented, allowing users to transfer files between local and remote computers seamlessly. This is particularly useful for users who need to access files on their remote computers from a different location.

Another possible feature that could be added to the remote desktop application is a chat feature. This would enable users to communicate with each other in real time while accessing their computers remotely, making it easier for remote teams to collaborate and share information. Additionally, screen-sharing functionality could be added to allow multiple users to view the same remote desktop session simultaneously.

Overall, the Remote Desktop application is a powerful tool that offers a wide range of features to facilitate remote access and control of computers. With its reliable communication, secure data transmission, and user-friendly interface, it is an ideal solution for users who need to access their files remotely.

**CODE**

**Server.py:**

import socket

import ssl

import subprocess

import sys

import os

IP = "127.0.0.1"

PORT = 5000

# Define a list of authorized usernames and passwords

AUTHORIZED\_USERS = {"user1": "password1", "user2": "password2"}

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind((IP, PORT))

server.listen(5)

print('Server is listening for connections...')

def authenticate(client\_socket):

    while True:

        username = client\_socket.recv(1024).decode().strip()

        password = client\_socket.recv(1024).decode().strip()

        if username in AUTHORIZED\_USERS and AUTHORIZED\_USERS[username] == password:

            client\_socket.sendall("\nAuthentication successful".encode())

            return True

        else:

            client\_socket.sendall("\nInvalid username or password. Please try again.".encode())

def receive\_file(client\_socket, filename):

    with open(filename, "wb") as f:

        while True:

            data = client\_socket.recv(1024)

            if not data:

                break

            f.write(data)

def send\_file(client\_socket, filename):

    with open(filename, "rb") as f:

        while True:

            data = f.read(1024)

            if not data:

                break

            client\_socket.sendall(data)

    client\_socket.sendall("file\_sent".encode())

def change\_directory(client\_socket, command):

    directory = command.split(maxsplit=1)

    if len(directory) > 1:

        directory = directory[1]

        try:

            os.chdir(directory)

            response = f"Changed directory to {directory}"

        except Exception as e:

            response = f"Failed to change directory: {str(e)}"

    else:

        directory = os.getcwd()

        response = f"Current directory: {directory}"

    client\_socket.sendall(response.encode())

def list\_directory(client\_socket):

    try:

        files = os.listdir('.')

        response = "\n".join(files)

    except Exception as e:

        response = f"Failed to list directory: {str(e)}"

    client\_socket.sendall(response.encode())

def create\_file(client\_socket, filename):

    try:

        open(filename, 'a').close()

        response = f"Created file: {filename}"

    except Exception as e:

        response = f"Failed to create file: {str(e)}"

    client\_socket.sendall(response.encode())

def delete\_file(client\_socket, filename):

    try:

        os.remove(filename)

        response = f"Deleted file: {filename}"

    except Exception as e:

        response = f"Failed to delete file: {str(e)}"

    client\_socket.sendall(response.encode())

def rename\_file(client\_socket, command):

    filenames = command.split(maxsplit=2)

    if len(filenames) == 3:

        old\_name = filenames[1]

        new\_name = filenames[2]

        try:

            os.rename(old\_name, new\_name)

            response = f"Renamed file {old\_name} to {new\_name}"

        except Exception as e:

            response = f"Failed to rename file: {str(e)}"

    else:

        response = "Invalid syntax. Usage: rename old\_name new\_name"

    client\_socket.sendall(response.encode())

def handle\_command(client\_socket, command):

    commands = {

    "quit": lambda: sys.exit(0),

    "send\_file": lambda: receive\_file(client\_socket, command.split()[1]),

    "receive\_file": lambda: send\_file(client\_socket, command.split()[1]),

    "cd": lambda: change\_directory(client\_socket, command),

    "ls": lambda: list\_directory(client\_socket),

    "create\_file": lambda: create\_file(client\_socket, command.split()[1]),

    "delete\_file": lambda: delete\_file(client\_socket, command.split()[1]),

    "rename\_file": lambda: rename\_file(client\_socket, command),

    }

    if command.split()[0] in commands:

        commands[command.split()[0]]()

    else:

        try:

            output = subprocess.check\_output(command, shell=True, stderr=subprocess.STDOUT)

        except subprocess.CalledProcessError as e:

            output = e.output

error\_message = f"Error: Unrecognized command '{command.split()[0]}'.\n{output.decode()}"

        client\_socket.sendall(error\_message.encode())

while True:

    client\_socket, client\_address = server.accept()

    print(f"Received connection from {client\_address}")

    # Authenticate the client

    if not authenticate(client\_socket):

        client\_socket.close()

        continue

    # Receive and execute commands from the client

    while True:

        command = client\_socket.recv(1024).decode().strip()

        if not command:

            break

        handle\_command(client\_socket, command)

    # Close the connection

    client\_socket.close()

server.close()

**Client.py:**

import socket

import ssl

import subprocess

import sys

IP = "127.0.0.1"

PORT = 5000

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect((IP, PORT))

# Function to authenticate the client

def authenticate():

    while True:

        username = input("Enter username: ")

        password = input("Enter password: ")

        client.sendall(username.encode())

        client.sendall(password.encode())

        response = client.recv(1024).decode()

        print(response)

        if response == "\nAuthentication successful":

            return True

        else:

            continue

# Authenticate the client before allowing commands

authenticated = authenticate()

if not authenticated:

    client.close()

    sys.exit(0)

def send\_file(filename):

    client.sendall(f"send\_file {filename}".encode())

    with open(filename, "rb") as f:

        while True:

            data = f.read(1024)

            if not data:

                break

            client.sendall(data)

    client.sendall("file\_sent".encode())

def receive\_file(filename):

    client.sendall(f"receive\_file {filename}".encode())

    with open(filename, "wb") as f:

        while True:

            data = client.recv(1024)

            if not data:

                break

            if data == b"file\_sent":

                break

            f.write(data)

while True:

    try:

        command = input("Enter a command (or type 'quit' to exit): ")

        if not command:

            continue

        if command == "quit":

            client.sendall(command.encode())

            client.close()

            sys.exit(0)

        elif command == "send\_file":

            filename = input("Enter the filename: ")

            send\_file(filename)

        elif command == "receive\_file":

            filename = input("Enter the filename: ")

            receive\_file(filename)

        elif command.startswith("cd"):

            client.sendall(command.encode())

            response = client.recv(1024).decode()

            print(response)

        else:

            client.sendall(command.encode())

            result = client.recv(4096).decode()

            print(result)

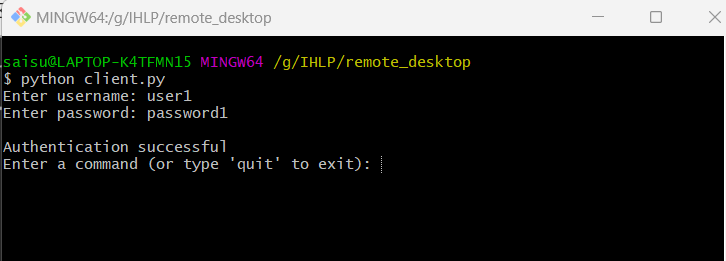
    except Exception as e:

        print(f"Error: {str(e)}")

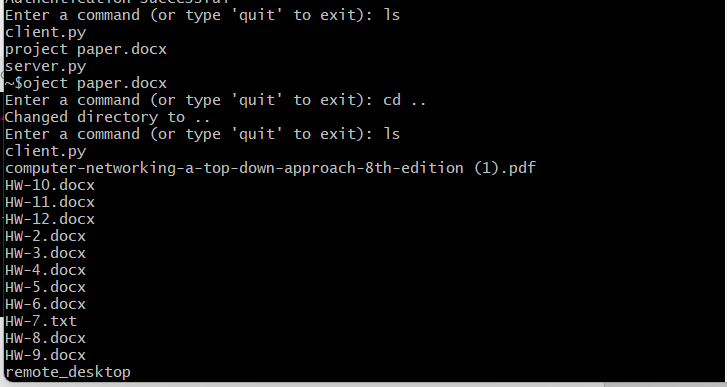
client.close()

**OUTPUT SCREENSHOTS**

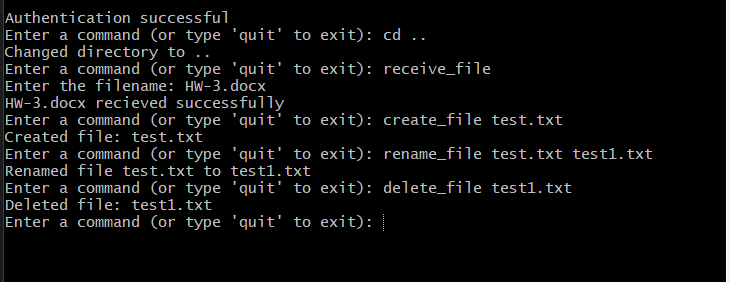
**Authentication:**

****

**Remote File Access:**

****

**File Transfer: (Create, Delete, Rename, send/receive)**

****

**CONCLUSION**

In summary, the remote desktop application developed using socket programming is an efficient and reliable tool for remote access and control of a computer over a network. Socket programming provides a robust method of communication between the client and server, ensuring that all actions performed by the client are executed on the server in real time.

The client-server architecture used in the application enables clients to connect to the server and control a computer remotely. To ensure the security of the application, user authentication has been implemented, ensuring that only authorized users can connect to the server. The file transfer feature allows users to transfer files between local and remote computers, making the application more versatile and useful for different purposes.

In the future, additional features such as encryption and multi-factor authentication could be added to enhance the security of the application further. Additionally, the application could be extended to support multiple platforms, such as mobile devices, making it more accessible and versatile.

Overall, the remote desktop application developed using socket programming provides an efficient and reliable method of remotely accessing and controlling a computer over a network. With the addition of various features, the application can be customized to meet the specific needs of different users.