CONCURRENT FILE SERVER

Aim

To develop a concurrent file server which will provide the file requested by client if it exists. If not server sends appropriate message to the client. Server should also send its process ID (PID) to clients for display along with file or the message.

Theory

Server - A server is a software that waits for client requests and serves or processes them accordingly.

Client - A client is requester of this service. A client program request for some resources to the server and server responds to that request.

Socket - It is the endpoint of a bidirectional communication channel between a server and a client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

FTP (File Transfer Protocol) is a client-server protocol that may be used to transfer files between computers on the internet. The client asks for the files and the server provides them. It is a standard network protocol used for the transfer of computer files between a client and server on a computer network. FTP is built on a client-server model architecture and uses separate control and data connections between the client and the server.FTP users may authenticate themselves with a clear text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. FTP sessions work in passive or active modes. In active mode, after a client initiates a session via a command channel request, the server initiates a data connection back to the client and begins transferring data. In passive mode, the server instead uses the command channel to send the client the information it needs to open a data channel. Because passive mode has the client initiating all connections, it works well across firewalls and Network Address Translation (NAT) gateways.

Algorithm

Algorithm for client

1. START
2. Create the socket using the function socket()
3. Configure socket details
4. Connect the socket to server using function connect()
5. Send the name of the file to search for to the server
6. If found,receive the file
7. If not found ,console out the appropriate message .
8. Receive the server’s PID .
9. Terminate connection

10. Stop

Algorithm for server

1. START
2. Configure socket details
3. Bind the address struct to the socket using bind ()
4. Listen to the socket
5. A new socket for t he incoming connection is created using acept()
6. Recieve the name of the file to search for
7. Check for the file using access()
8. If the file is found , goto step 9 ,else goto step 10
9. Send the file to the client using sendfile()
10. Send the response ”NO” to the client
11. Send the PID to the client
12. STOP

Code

Server.py

```
import socket
from _thread import start_new_thread
import threading
from os import getpid

print_lock = threading.Lock()
pid = getpid()


def threaded(c):
    while True:
        try:
            data = c.recv(1024)
        except Exception:
            break

        if not data:
            break

        fileName = data.decode()
        print(f"filename: {fileName}")
        c.send(str(pid).encode())

        try:
            file = open(fileName, "rb")
        except FileNotFoundError:
            c.send("File Not Found")
            print("File Not Found")
            break

        inp = file.read(1024)
        while inp:
            c.send(inp)
            inp = file.read(1024)
        c.send("".encode())

        c.close()

    print(f"{c} disconnected")
```

```
    print_lock.release()


def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(("127.0.0.1", 9999))
    s.listen(5)
    print("Listening for connections")

    while True:

        c, addr = s.accept()

        print_lock.acquire()
        print("Connected to :", addr)

        start_new_thread(threaded, (c,))
    s.close()


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

Client.py

```
import socket


def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(("127.0.0.1", 5001))

    fileName = input("File name: ")
    # fileName = "test-file.txt"

    outputFile = open("received.txt", "wb")
    s.send(fileName.encode())
    pid = s.recv(1024)
    print(f"PID of server process: {pid.decode()}")
    data = s.recv(1024)
    while data:
        if not data.decode():
            s.shutdown(socket.SHUT_WR)
            break
        # print(data)
        outputFile.write(data)
        data = s.recv(1024)

    s.close()
    print(f"{fileName} received.")


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

Output

```
[sachin@sachin ~]$ python3 fserver.py
Listening for connections
Connected to : ('127.0.0.1', 58532)
filename: a.txt
<socket.socket [closed] fd=-1, family=AddressFamily.AF_INET, type=SocketKind.SOC
K_STREAM, proto=0> disconnected
Connected to : ('127.0.0.1', 58534)
filename: a.txt
<socket.socket [closed] fd=-1, family=AddressFamily.AF_INET, type=SocketKind.SOC
K_STREAM, proto=0> disconnected
```

```
[sachin@sachin ~]$ python3 fclient.py
File name: a.txt
PID of server process: 8402
a.txt received.
[sachin@sachin ~]$ cat received.txt
data being sent gets stored in recieved.txt
[sachin@sachin ~]$ cat a.txt
data being sent gets stored in recieved.txt
```

Result

Develop a concurrent file server which will provide the file requested by client if it exists& server sends appropriate message to the client if it doesn’t exist.