CS 461

Lab Assignment 7

Name: Gandhi Dhruv Vipulkumar

Institute ID: 202151053

Date: 21-10-2024

**Q. Implement Distributed File System Application**

**Master\_server.py**

import socket  
import threading  
import pickle  
  
# Master server to manage metadata and coordinate between nodes  
class MasterServer:  
    def \_\_init\_\_(self, host='localhost', port=5000):  
        self.host = host  
        self.port = port  
        self.metadata = {}  # Stores file -> node mapping  
        self.nodes = {}  # Stores node info (node\_id -> address)  
      
    def start(self):  
        server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
        server.bind((self.host, self.port))  
        server.listen(5)  
        print("Master server started on port", self.port)  
          
        while True:  
            client\_socket, client\_address = server.accept()  
            threading.Thread(target=self.handle\_client, args=(client\_socket,)).start()  
      
    def handle\_client(self, client\_socket):  
        try:  
            request = client\_socket.recv(1024)  
            request = pickle.loads(request)  
              
            if request['type'] == 'register\_node':  
                # Register a new data node  
                node\_id = request['node\_id']  
                node\_address = request['address']  
                self.nodes[node\_id] = node\_address  
                client\_socket.send(b"Node registered successfully.")  
              
            elif request['type'] == 'upload':  
                # Handle file upload from client  
                filename = request['filename']  
                file\_data = request['data']  
                # Distribute file to nodes (simple round-robin for now)  
                node\_id = list(self.nodes.keys())[0]  # Simplified: always use first node  
                node\_address = self.nodes[node\_id]  
                self.metadata[filename] = node\_id  
                  
                # Send file to node  
                self.send\_file\_to\_node(node\_address, filename, file\_data)  
                client\_socket.send(b"File uploaded successfully.")  
              
            elif request['type'] == 'download':  
                # Handle file download request from client  
                filename = request['filename']  
                if filename in self.metadata:  
                    node\_id = self.metadata[filename]  
                    node\_address = self.nodes[node\_id]  
                    file\_data = self.get\_file\_from\_node(node\_address, filename)  
                    client\_socket.send(pickle.dumps({'status': 'success', 'data': file\_data}))  
                else:  
                    client\_socket.send(pickle.dumps({'status': 'error', 'message': 'File not found.'}))  
              
        except Exception as e:  
            print("Error:", e)  
        finally:  
            client\_socket.close()  
      
    def send\_file\_to\_node(self, node\_address, filename, file\_data):  
        # Function to send file to a specific node  
        node\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
        node\_socket.connect(node\_address)  
        request = pickle.dumps({'type': 'store', 'filename': filename, 'data': file\_data})  
        node\_socket.send(request)  
        node\_socket.close()  
      
    def get\_file\_from\_node(self, node\_address, filename):  
        # Function to retrieve file from a node  
        node\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
        node\_socket.connect(node\_address)  
        request = pickle.dumps({'type': 'retrieve', 'filename': filename})  
        node\_socket.send(request)  
        response = node\_socket.recv(1024)  
        node\_socket.close()  
        return pickle.loads(response)['data']  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    master\_server = MasterServer()  
    master\_server.start()

**data\_node.py**

import socket  
import threading  
import pickle  
import os  
  
# Data node that stores file chunks  
  
class DataNode:  
    def \_\_init\_\_(self, node\_id, host='localhost', port=8000, master\_host='localhost', master\_port=5000):  
        self.node\_id = node\_id  
        self.host = host  
        self.port = port  
        self.master\_address = (master\_host, master\_port)  
        self.storage = {}  # Dictionary to store files  
  
    def start(self):  
        # Register with the master server  
        self.register\_with\_master()  
  
        # Start listening for requests from master or clients  
        server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
        server.bind((self.host, self.port))  
        server.listen(5)  
        print(f"Data node {self.node\_id} started on port", self.port)  
  
        while True:  
            client\_socket, client\_address = server.accept()  
            threading.Thread(target=self.handle\_request,  
                             args=(client\_socket,)).start()  
  
    def handle\_request(self, client\_socket):  
        try:  
            request = client\_socket.recv(1024)  
            request = pickle.loads(request)  
  
            if request['type'] == 'store':  
                # Store file data  
                filename = request['filename']  
                file\_data = request['data']  
                self.storage[filename] = file\_data  
                print(f"Stored file {filename}")  
  
            elif request['type'] == 'retrieve':  
                # Retrieve file data  
                filename = request['filename']  
                if filename in self.storage:  
                    client\_socket.send(pickle.dumps(  
                        {'status': 'success', 'data': self.storage[filename]}))  
                else:  
                    client\_socket.send(pickle.dumps(  
                        {'status': 'error', 'message': 'File not found.'}))  
  
        except Exception as e:  
            print("Error:", e)  
        finally:  
            client\_socket.close()  
  
    def register\_with\_master(self):  
        node\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
        node\_socket.connect(self.master\_address)  
        request = pickle.dumps(  
            {'type': 'register\_node', 'node\_id': self.node\_id, 'address': (self.host, self.port)})  
        node\_socket.send(request)  
        node\_socket.close()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    data\_node = DataNode(node\_id=3)  
    data\_node.start()

**client.py**

import socket  
import pickle  
import os  
  
# Client for interacting with the DFS  
  
class DFSClient:  
    def \_\_init\_\_(self, master\_host='localhost', master\_port=5000):  
        self.master\_address = (master\_host, master\_port)  
  
    def upload(self, filename):  
        if not os.path.exists(filename):  
            print(f"File '{filename}' does not exist.")  
            return  
  
        try:  
            with open(filename, 'rb') as file:  
                file\_data = file.read()  
  
            client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
            client\_socket.connect(self.master\_address)  
  
            request = pickle.dumps({  
                'type': 'upload',  
                'filename': filename,  
                'data': file\_data  
            })  
            client\_socket.send(request)  
  
            response = client\_socket.recv(1024)  
            print(response.decode())  
        except Exception as e:  
            print(f"Error during upload: {e}")  
        finally:  
            client\_socket.close()  
  
    def download(self, filename):  
        try:  
            client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
            client\_socket.connect(self.master\_address)  
  
            request = pickle.dumps({'type': 'download', 'filename': filename})  
            client\_socket.send(request)  
  
            response = client\_socket.recv(1024)  
            response = pickle.loads(response)  
  
            if response['status'] == 'success':  
                file\_data = response['data']  
                with open(filename, 'wb') as file:  
                    file.write(file\_data)  
                print(f"Downloaded and saved file: {filename}")  
            else:  
                print(f"Error: {response['message']}")  
        except Exception as e:  
            print(f"Error during download: {e}")  
        finally:  
            client\_socket.close()  
  
def main():  
    client = DFSClient()  
  
    while True:  
        print("\nDistributed File System Client")  
        print("1. Upload a file")  
        print("2. Download a file")  
        print("3. Exit")  
        choice = input("Choose an option: ")  
  
        if choice == '1':  
            filename = input("Enter the filename to upload: ")  
            client.upload(filename)  
        elif choice == '2':  
            filename = input("Enter the filename to download: ")  
            client.download(filename)  
        elif choice == '3':  
            print("Exiting client.")  
            break  
        else:  
            print("Invalid option. Please choose 1, 2, or 3.")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    main()

**Code Explanation:**

**1. Set up the Master Server:** The master server is responsible for maintaining the metadata, mapping files to nodes, and coordinating file uploads/downloads.

**2. Set up Data Nodes:** The data nodes are responsible for storing file chunks and responding to requests from the master server.

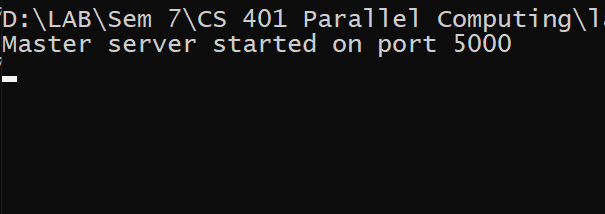
**3. Client Implementation:** Clients interact with the DFS by uploading or downloading files through the master server.

**Key Features:**

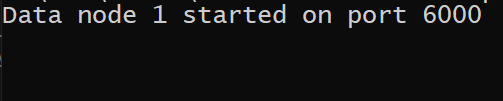
* **File Upload and Download**: Clients can upload files, which are then distributed to data nodes by the master server. Files can be downloaded by requesting from the master.
* **Simple Round-Robin Distribution**: Files are stored on different nodes in a round-robin manner.
* **Node Registration**: Data nodes register with the master server upon startup, allowing for dynamic node management.

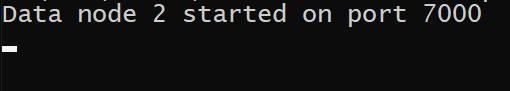
**Testing Phase:**

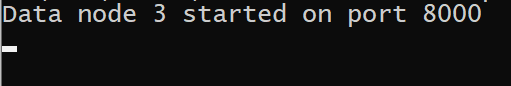
**1) Start master server on port 5000**

****

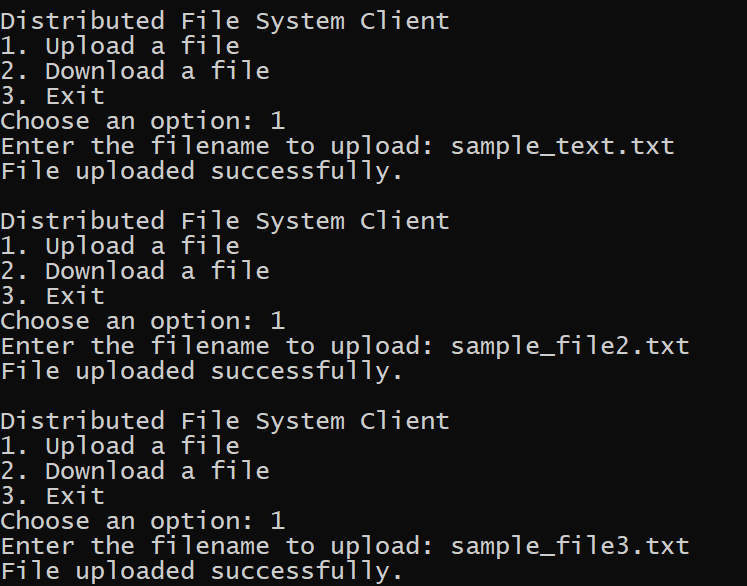
**2) Start 3 data nodes with id=1, 2, 3 on port 6000, 7000 and 8000 respectively**

****

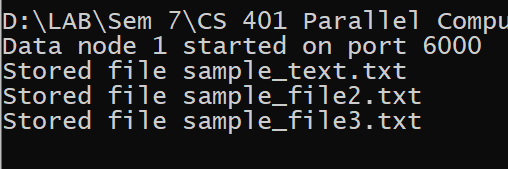
****

****

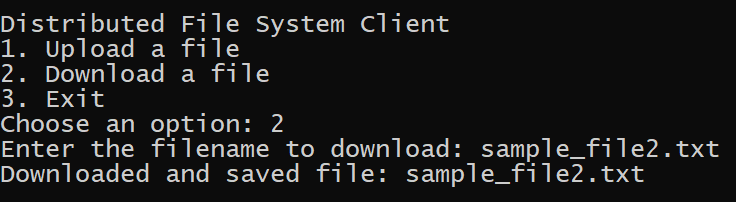
**3) Start client and upload sample\_text.txt, sample\_file2.txt and sample\_file3.txt**

****

**4) All the files uploaded on nodeid=1 on port 6000**

****

**5) One can also download the file uploaded on server**

****

**Conclusion:** Successfully implemented distributed file system application in python.