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

```python
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()
```

```python
if __name__ == '__main__':
    data_node = DataNode(node_id=3)
    data_node.start()
```

client.py

```python
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.")
```

```python
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.

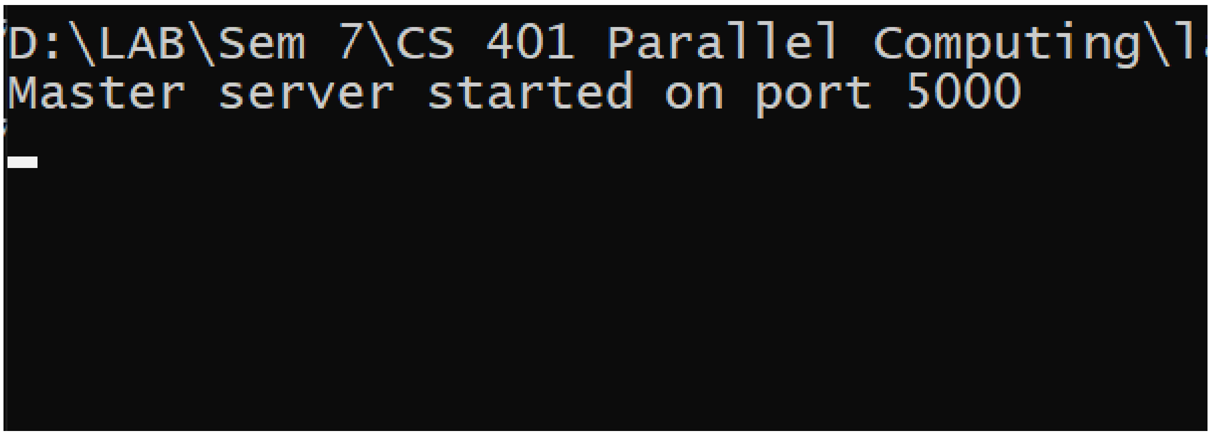
3. Client Implementation: Clients interact with the DFS by uploading or downloading files through the master server.

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.

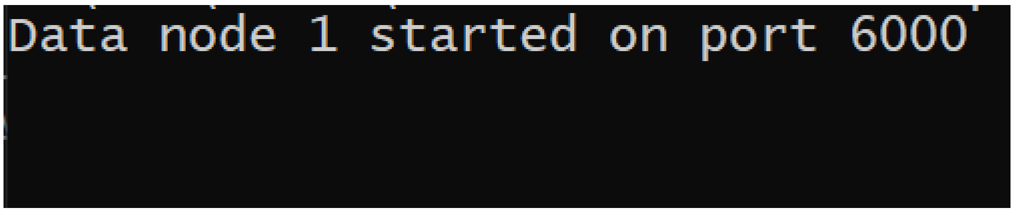
Testing Phase:

1) Start master server on port 5000

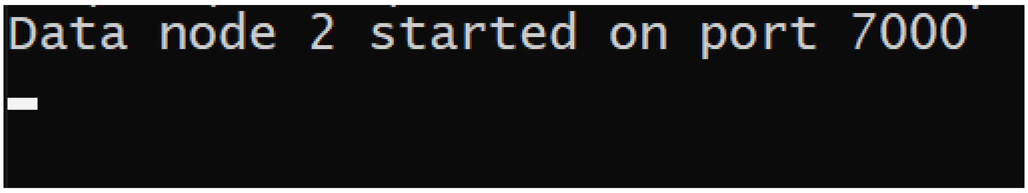


```
D:\LAB\Sem 7\CS 401 Parallel Computing\l
Master server started on port 5000
```

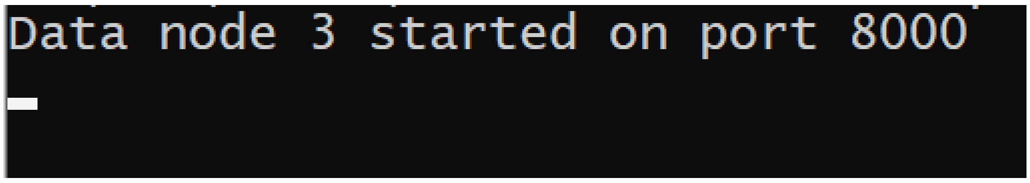
2) Start 3 data nodes with id=1, 2, 3 on port 6000, 7000 and 8000 respectively



```
Data node 1 started on port 6000
```

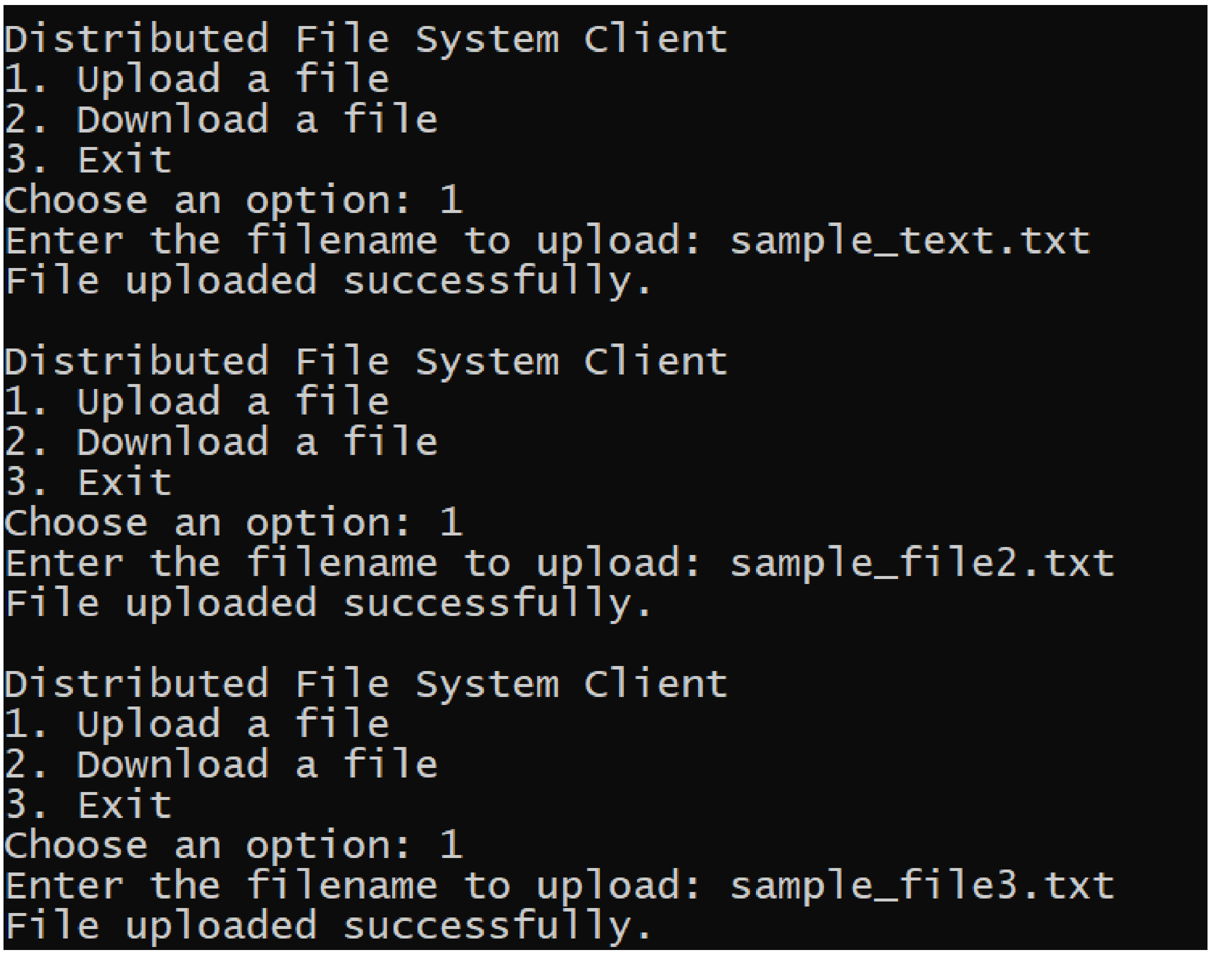


```
Data node 2 started on port 7000
```



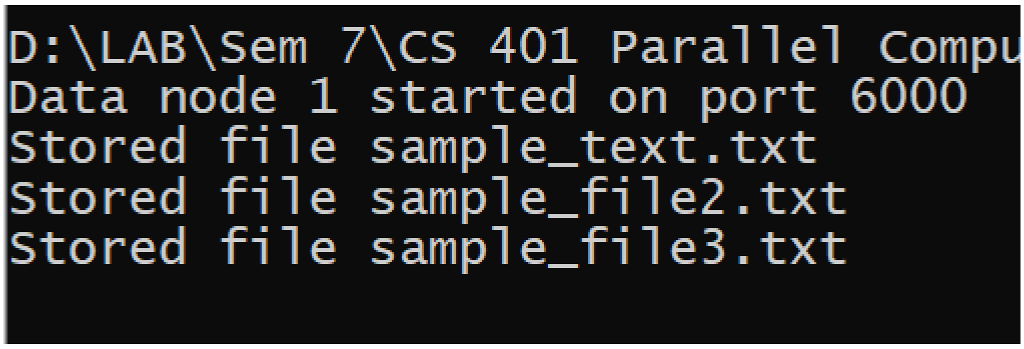
```
Data node 3 started on port 8000
```

3) Start client and upload sample_text.txt, sample_file2.txt and sample_file3.txt



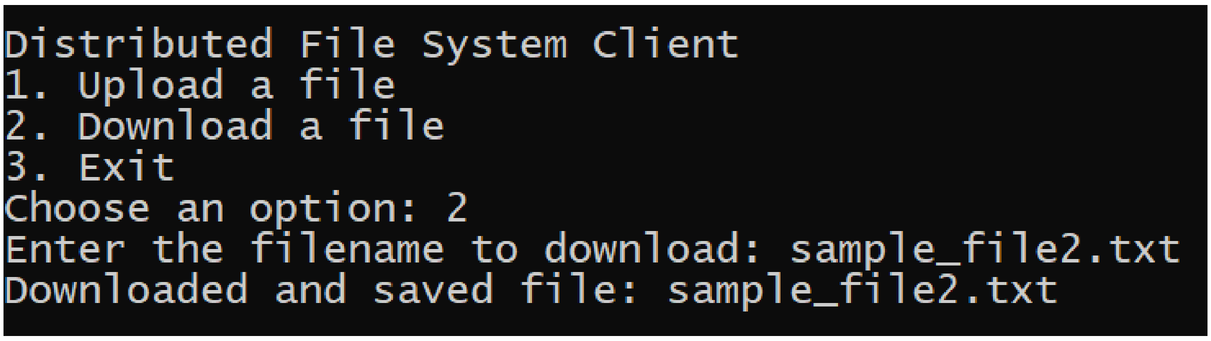
```
Distributed File System Client
1. Upload a file
2. Download a file
3. Exit
Choose an option: 1
Enter the filename to upload: sample_text.txt
File uploaded successfully.

Distributed File System Client
1. Upload a file
2. Download a file
3. Exit
Choose an option: 1
Enter the filename to upload: sample_file2.txt
File uploaded successfully.

Distributed File System Client
1. Upload a file
2. Download a file
3. Exit
Choose an option: 1
Enter the filename to upload: sample_file3.txt
File uploaded successfully.
```

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



```
D:\LAB\Sem 7\CS 401 Parallel Compu
Data node 1 started on port 6000
Stored file sample_text.txt
Stored file sample_file2.txt
Stored file sample_file3.txt
```

5) One can also download the file uploaded on server



```
Distributed File System Client
1. Upload a file
2. Download a file
3. Exit
Choose an option: 2
Enter the filename to download: sample_file2.txt
Downloaded and saved file: sample_file2.txt
```

Conclusion: Successfully implemented distributed file system application in python.