Distributed Systems

Challenging Task -1
Implementation of Distributed Shared Memory Using Client–Server Architecture

Roll. No: CH.SC.U4CSE23224
Name: Manoj Mehra V

1.Demonstrate the concept of Distributed Shared Memory in a distributed system by implementing a client–server application where multiple computers with different IP addresses can read and update a shared variable over a network.

Objective of the Exercise:

To design and implement a distributed shared memory system that allows multiple nodes to access, modify, and synchronize a common data value using network communication.

Program Code:

i) server.py:

server.py client.py

C: > Users > mehra > Desktop > server.py > WriterPriorityRWLock > __init__

```python
import socket
import struct
import threading
import mmap


class WriterPriorityRWLock:
    def __init__(self):
        self.readers = 0
        self.writers_waiting = 0
        self.writer_active = False
        self.lock = threading.Lock()
        self.ok_to_read = threading.Condition(self.lock)
        self.ok_to_write = threading.Condition(self.lock)

    def acquire_read(self):
        with self.lock:
            while self.writer_active or self.writers_waiting > 0:
                self.ok_to_read.wait()
            self.readers += 1

```

server.py client.py

C: > Users > mehra > Desktop > server.py > WriterPriorityRWLock > __init__

```
7  class WriterPriorityRWLock:
21
22      def release_read(self):
23          with self.lock:
24              self.readers -= 1
25              if self.readers == 0:
26                  self.ok_to_write.notify()
27
28      def acquire_write(self):
29          with self.lock:
30              self.writers_waiting += 1
31              while self.readers > 0 or self.writer_active:
32                  self.ok_to_write.wait()
33              self.writers_waiting -= 1
34              self.writer_active = True
35
36      def release_write(self):
37          with self.lock:
38              self.writer_active = False
39              if self.writers_waiting > 0:
40                  self.ok_to_write.notify()
41              else:
42                  self.ok_to_read.notify_all()
43
```

server.py client.py

C: > Users > mehra > Desktop > server.py > WriterPriorityRWLock > __init__

```
45  def create_shared_memory():
46      shm = mmap.mmap(-1, 4)
47      shm.write(struct.pack("i", 0))
48      shm.seek(0)
49      return shm
50
51
52  rwlock = WriterPriorityRWLock()
53
54
55  def handle_client(conn, addr, shared_memory):
56      print(f"[+] Connected: {addr}")
57      while True:
58          cmd = conn.recv(1)
59          if not cmd:
60              break
61
62          if cmd == b"R":
63              rwlock.acquire_read()
64              shared_memory.seek(0)
65              value = shared_memory.read(4)
66              rwlock.release_read()
67              conn.sendall(value)
68
69          elif cmd == b"W":
70              data = conn.recv(4)
71              if not data:
72                  break
```

server.py client.py

C: > Users > mehra > Desktop > server.py > WriterPriorityRWLock > __init__

```
55  def handle_client(conn, addr, shared_memory):
73                  rwlock.acquire_write()
74                  shared_memory.seek(0)
75                  shared_memory.write(data)
76                  rwlock.release_write()
77                  conn.sendall(b"OK")
78
79      conn.close()
80      print(f"[-] Disconnected: {addr}")
81
82
83  def server():
84      shared_memory = create_shared_memory()
85      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
86      s.bind(("0.0.0.0", 12345))
87      s.listen(10)
88      print("Server listening on port 12345...")
89
90      while True:
91          conn, addr = s.accept()
92          threading.Thread(
93              target=handle_client, args=(conn, addr, shared_memory), daemon=True
94          ).start()
95
96
97  if __name__ == "__main__":
98      server()
99
```

ii)client.py:

server.py client.py

C: > Users > mehra > Desktop > client.py > ...

```
1   import socket
2   import struct
3
4
5   def client(server_ip):
6       s = socket.socket()
7       s.connect((server_ip, 12345))
8
9       print("Connected to server.")
10      print("R -> Read")
11      print("W -> Write")
12      print("Q -> Quit")
13
14      while True:
15          cmd = input("\nEnter command (R/W/Q): ").strip().upper()
16
17          if cmd == "Q":
18              break
19
20          if cmd == "R":
21              s.sendall(b"R")
22              data = s.recv(4)
23              value = struct.unpack("i", data)[0]
24              print("Shared Value:", value)
25
```

server.py client.py

C: > Users > mehra > Desktop > client.py > ...

```
def client(server_ip):

        elif cmd == "W":
            value = int(input("Enter new value: "))
            s.sendall(b"W")
            s.sendall(struct.pack("i", value))
            print("Server:", s.recv(2).decode())

        else:
            print("Invalid command")

    s.close()


if __name__ == "__main__":
    client("11.12.7.152")  # Replace with server IP
```

Input(Client side):

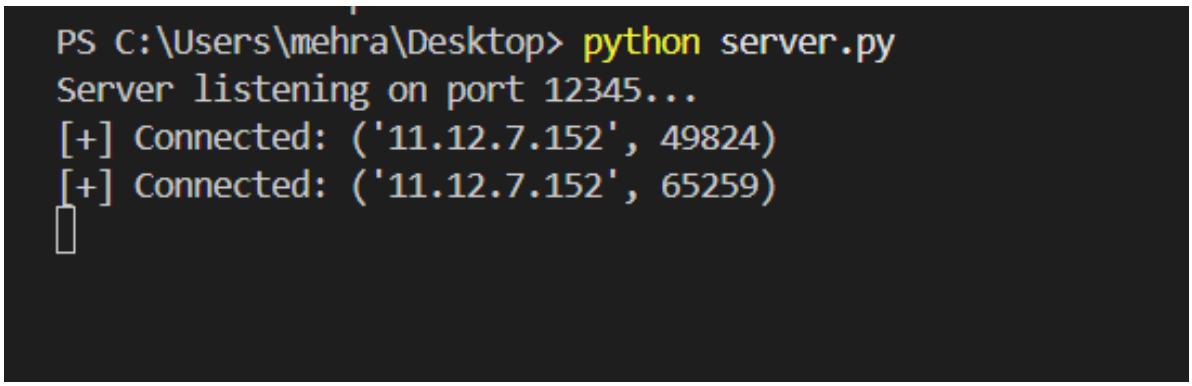
```
W
Enter new value: 20
```

Output:

```
Server: OK
```

Screenshots of Output:

Server:



Client1:

```
PS C:\Users\mehra\Desktop> python client.py
Connected to server.
Connected to server.
R -> Read
W -> Write
R -> Read
W -> Write
W -> Write
Q -> Quit
Q -> Quit

Enter command (R/W/Q): W
Enter new value: 10
Server: OK

Enter command (R/W/Q): r
Shared Value: 25
```

Client2:

```
PS C:\Users\mehra\Desktop> python client.py
Connected to server.
Connected to server.
R -> Read
R -> Read
W -> Write
W -> Write
Q -> Quit
Q -> Quit

Enter command (R/W/Q): R
Shared Value: 10

Enter command (R/W/Q): w
Enter new value: 25
Server: OK

Enter command (R/W/Q):
```

Inference:

The experiment proves that a shared variable can be accessed and modified by multiple computers with different IP addresses through a centralized server. Any update made by one client is immediately reflected to all other clients, demonstrating the working of distributed shared memory with proper synchronization and consistency control.

--x--