DC PRACTICAL – COMPLETE CODE WITH INPUT & OUTPUT

1. Load Balancing using Round Robin Algorithm

Code:
```
class RoundRobin:
    def __init__(self, servers):
        self.servers = servers
        self.index = 0

    def next_server(self):
        if not self.servers:
            return None
        server = self.servers[self.index]
        self.index = (self.index + 1) % len(self.servers)
        return server

servers = input("Enter servers (comma separated): ").split(",")
rr = RoundRobin(servers)

n = int(input("Enter number of requests: "))

for r in range(1, n+1):
    print(f"Request {r} served by → {rr.next_server()}")
```

Input:
```
Enter servers (comma separated): S1,S2,S3
Enter number of requests: 6
```

Output:
```
Request 1 served by → S1
Request 2 served by → S2
Request 3 served by → S3
Request 4 served by → S1
Request 5 served by → S2
Request 6 served by → S3
```

----

2. Scalability using Multithreading

Code:
```
import threading
import queue
import time

tasks = queue.Queue()

for i in range(5):
    tasks.put(f"Task {i}")

def worker(id):
    while not tasks.empty():
        try:
            t = tasks.get_nowait()
        except:
            break
        print(f"Worker {id} processing {t}")
        time.sleep(0.5)
        print(f"Worker {id} finished {t}")
        tasks.task_done()

threads = []

for i in range(2):
    t = threading.Thread(target=worker, args=(i,))
    threads.append(t)
    t.start()

for t in threads:
    t.join()
print("All tasks completed.")
```

```
Output:
Worker 0 processing Task 0
Worker 1 processing Task 1
...
All tasks completed.

------------------------------------------------------------
```

3. RPC Client/Server Communication

```
server.py:
import socket
import struct

server = socket.socket()
server.bind(("localhost", 6666))
server.listen(1)

print("Server started... Waiting for client...")
conn, addr = server.accept()
print("Client connected:", addr)

data = conn.recv(8)
a, b = struct.unpack('ii', data)

print("Sum =", a + b)

conn.close()
server.close()

client.py:
import socket
import struct

s = socket.socket()
s.connect(("localhost", 6666))

a = int(input("Enter 1st number: "))
b = int(input("Enter 2nd number: "))

s.send(struct.pack('ii', a, b))
s.close()

Client Input:
Enter 1st number: 10
Enter 2nd number: 15

Server Output:
Sum = 25

------------------------------------------------------------
```

4. Election Algorithm (Ring)

```
Code:
class Process:
    def __init__(self, pid):
        self.pid = pid

def ring_election(processes, start):
    msg = []
    i = start
    n = len(processes)

    print(f"Process {start} starts election")
    while True:
        msg.append(processes[i].pid)
        print("Passing →", msg)
        i = (i + 1) % n
        if i == start:
            break
    print("Coordinator =", max(msg))
```

```
Input:
Enter process IDs: 3,9,4,7,2
Enter initiator index: 2
```

Output:

```
Coordinator = 9
```

----

5. Distributed Deadlock Detection

Code:

```
resources = {
    "R1": "P1",
    "R2": "P1"
}

print("P1 acquired R1 and R2")
print("P2 waiting for R1")

if list(resources.values()).count("P1") == len(resources):
    print("Deadlock detected! P1 holds all resources.")
```

Output:

```
Deadlock detected! P1 holds all resources.
```

----

6. Name Resolution Protocol (DNS Simulation)

Code:

```
class DNS:
    def __init__(self):
        self.records = {
            "google.com": "8.8.8.8",
            "openai.com": "4.4.4.4"
        }
        self.cache = {}

    def resolve(self, host):
        if host in self.cache:
            return self.cache[host]
        ip = self.records.get(host)
        if ip:
            self.cache[host] = ip
        return ip

dns = DNS()

for h in ["google.com", "openai.com", "facebook.com"]:
    ip = dns.resolve(h)
    print(h, "→", ip)
```

Output:

```
google.com → 8.8.8.8
openai.com → 4.4.4.4
facebook.com → None
```

----

7. Mutual Exclusion Algorithm

Code:

```
lock = False
pid = 0

while True:
    key = input("Press any key to create new process (q to quit): ")
    if key == "q":
        break

    print(f"Process {pid} requesting access...")
```

```
if not lock:
    lock = True
    print(f"Process {pid} entered critical section")
    print(f"Process {pid} exited critical section")
    lock = False
else:
    print("Critical section busy!")

pid += 1
```