Name: Parkale Shreya Jagdish

Roll No.: 2447060

Batch : C

Problem Statement: Write code to simulate requests coming from clients and distribute them among the servers using the load balancing algorithms

In [1]:
```python
import random
from collections import defaultdict

# Simulating a Server
class Server:
    def __init__(self, name):
        self.name = name
        self.connections = 0  # For Least Connections algo
        self.total_requests = 0  # To track total requests handled

    def handle_request(self, request_id):
        self.connections += 1
        self.total_requests += 1
        print(f"Server {self.name} handled Request {request_id}")
        # After handling, decrease connection to simulate quick processing
        self.connections -= 1
```

In [2]:

```python
# Simulating Clients Sending Requests
def simulate_requests(load_balancer, algorithm, num_requests=10):

    print(f"\nSimulating {algorithm.upper()} Load Balancing:")

    for request_id in range(1, num_requests + 1):
        if algorithm == "round_robin":
            load_balancer.round_robin(request_id)
        elif algorithm == "random":
            load_balancer.random_choice(request_id)
        elif algorithm == "least_connections":
            load_balancer.least_connections(request_id)
        elif algorithm == "weighted_round_robin":
            load_balancer.weighted_round_robin(request_id)
        else:
            print("Unknown Algorithm!")

    # After Simulation, Print Request Count for Each Server
    print("\nRequest Distribution:")
    for server in load_balancer.servers:
        print(f"Server {server.name}: {server.total_requests} requests handled")
    print("-" * 50)
```

In [3]:

```python
# Load Balancer Class
class LoadBalancer:
    def __init__(self, servers, weights=None):
        self.servers = servers
        self.index = 0  # For Round Robin
        self.weights = weights
        if weights:
            self.weighted_servers = []
            for server, weight in zip(servers, weights):
                self.weighted_servers.extend([server] * weight)

    def round_robin(self, request_id):
        server = self.servers[self.index]
        server.handle_request(request_id)
        self.index = (self.index + 1) % len(self.servers)

    def random_choice(self, request_id):
        server = random.choice(self.servers)
        server.handle_request(request_id)

    def least_connections(self, request_id):
        server = min(self.servers, key=lambda s: s.connections)
        server.handle_request(request_id)

    def weighted_round_robin(self, request_id):
        if not self.weights:
            print("No weights provided!")
            return
        server = random.choice(self.weighted_servers)
        server.handle_request(request_id)
```

In [4]:

```python
if __name__ == "__main__":
    # Create some server instances
    servers1 = [Server("A"), Server("B"), Server("C")]
    servers2 = [Server("A"), Server("B"), Server("C")]
    servers3 = [Server("A"), Server("B"), Server("C")]
    servers4 = [Server("A"), Server("B"), Server("C")]

    # Define weights for Weighted Round Robin
    weights = [3, 1, 2]  # A gets 3 times more, B gets 1 time, C gets 2 times

    # Create LoadBalancer instances
    lb_round_robin = LoadBalancer(servers1)
    lb_random = LoadBalancer(servers2)
    lb_least_connections = LoadBalancer(servers3)
    lb_weighted = LoadBalancer(servers4, weights)

    # Simulate different algorithms
    simulate_requests(lb_round_robin, "round_robin")
    simulate_requests(lb_random, "random")
    simulate_requests(lb_least_connections, "least_connections")
    simulate_requests(lb_weighted, "weighted_round_robin")
```

```
Simulating ROUND_ROBIN Load Balancing:
Server A handled Request 1
Server B handled Request 2
Server C handled Request 3
Server A handled Request 4
Server B handled Request 5
Server C handled Request 6
Server A handled Request 7
Server B handled Request 8
Server C handled Request 9
Server A handled Request 10

Request Distribution:
Server A: 4 requests handled
Server B: 3 requests handled
Server C: 3 requests handled
-------------------------------------------------

Simulating RANDOM Load Balancing:
Server C handled Request 1
Server B handled Request 2
Server B handled Request 3
Server B handled Request 4
Server C handled Request 5
Server B handled Request 6
Server B handled Request 7
Server B handled Request 8
Server C handled Request 9
Server C handled Request 10

Request Distribution:
Server A: 0 requests handled
Server B: 6 requests handled
Server C: 4 requests handled
-------------------------------------------------

Simulating LEAST_CONNECTIONS Load Balancing:
Server A handled Request 1
Server A handled Request 2
Server A handled Request 3
Server A handled Request 4
Server A handled Request 5
```

```
Server A handled Request 6
Server A handled Request 7
Server A handled Request 8
Server A handled Request 9
Server A handled Request 10

Request Distribution:
Server A: 10 requests handled
Server B: 0 requests handled
Server C: 0 requests handled
--------------------------------------------------

Simulating WEIGHTED_ROUND_ROBIN Load Balancing:
Server A handled Request 1
Server A handled Request 2
Server A handled Request 3
Server C handled Request 4
Server C handled Request 5
Server B handled Request 6
Server B handled Request 7
Server C handled Request 8
Server A handled Request 9
Server A handled Request 10

Request Distribution:
Server A: 5 requests handled
Server B: 2 requests handled
Server C: 3 requests handled
--------------------------------------------------
```

In [ ]: