**Course:** BCA **Semester:** 4

**Q.1 Python Code to implement peer-to-peer using block chain?**

# OBJECTIVE

We'll set up a P2P network with two peers (peer1 and peer2).

 Peers will exchange and synchronize their blockchains.

#  We'll output the current blockchain data each time a new block is added or received.

CODE

import hashlib

import json

import time

import socket

from threading import Thread

import random

# Block Class to represent each block

class Block:

def \_\_init\_\_(self, index, previous\_hash, timestamp, data, hash):

self.index = index

self.previous\_hash = previous\_hash

self.timestamp = timestamp

self.data = data

self.hash = hash

# Blockchain Class to represent the chain of blocks

class Blockchain:

def \_\_init\_\_(self):

self.chain = []

self.create\_genesis\_block()

def create\_genesis\_block(self):

# Create the first block (genesis block)

genesis\_block = Block(0, "0", int(time.time()), "Genesis Block", self.calculate\_hash(0, "0", "Genesis Block"))

self.chain.append(genesis\_block)

def add\_block(self, data):

last\_block = self.chain[-1]

new\_block = self.create\_new\_block(data, last\_block)

self.chain.append(new\_block)

self.print\_blockchain()

def create\_new\_block(self, data, last\_block):

index = last\_block.index + 1

timestamp = int(time.time())

previous\_hash = last\_block.hash

hash = self.calculate\_hash(index, previous\_hash, data)

return Block(index, previous\_hash, timestamp, data, hash)

def calculate\_hash(self, index, previous\_hash, data):

block\_content = f'{index}{previous\_hash}{data}'

return hashlib.sha256(block\_content.encode('utf-8')).hexdigest()

def get\_last\_block(self):

return self.chain[-1]

def print\_blockchain(self):

# Print the blockchain's blocks

print("\nBlockchain:")

for block in self.chain:

print(f"Block {block.index} [Data: {block.data}, Hash: {block.hash}]")

print("-" \* 50)

# Peer class that represents a node in the P2P network

class Peer:

def \_\_init\_\_(self, host, port, blockchain):

self.host = host

self.port = port

self.blockchain = blockchain

self.peers = []

def start\_server(self):

# Start the server to listen for incoming connections

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind((self.host, self.port))

server.listen(5)

print(f"Listening for incoming connections on {self.host}:{self.port}")

while True:

conn, addr = server.accept()

print(f"Connection established with {addr}")

Thread(target=self.handle\_client, args=(conn,)).start()

def handle\_client(self, conn):

data = conn.recv(1024)

if data:

peer\_blockchain = json.loads(data.decode('utf-8'))

print("Received blockchain data from peer...")

self.sync\_blockchain(peer\_blockchain)

def sync\_blockchain(self, peer\_blockchain):

if len(peer\_blockchain) > len(self.blockchain.chain):

print(f"Synchronizing blockchain with the peer. Peer chain length: {len(peer\_blockchain)}, Local chain length: {len(self.blockchain.chain)}")

self.blockchain.chain = peer\_blockchain

self.blockchain.print\_blockchain()

else:

print("Peer blockchain is not longer. No update needed.")

def broadcast\_block(self, new\_block):

for peer in self.peers:

peer.send\_block(new\_block)

def send\_block(self, new\_block):

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect((self.host, self.port))

client.send(json.dumps(new\_block).encode('utf-8'))

client.close()

def add\_peer(self, peer):

self.peers.append(peer)

# Main function to initialize blockchain and P2P network

def main():

blockchain = Blockchain()

# Initialize peers with unique ports

peer1 = Peer("127.0.0.1", 5000, blockchain)

peer2 = Peer("127.0.0.1", 5001, blockchain)

# Add some blocks (transactions)

blockchain.add\_block("Transaction 1")

blockchain.add\_block("Transaction 2")

# Start P2P server for each peer

Thread(target=peer1.start\_server).start()

Thread(target=peer2.start\_server).start()

# Simulate adding a block on peer1 and broadcasting it to peer2

new\_block = {

"index": 3,

"previous\_hash": blockchain.get\_last\_block().hash,

"timestamp": int(time.time()),

"data": "Transaction 3",

"hash": blockchain.calculate\_hash(3, blockchain.get\_last\_block().hash, "Transaction 3")

}

print("\n--- Broadcasting new block from peer1 ---")

peer1.broadcast\_block(new\_block)

# Add some delay before the next transaction

time.sleep(1)

# Simulate adding another block and broadcasting it

blockchain.add\_block("Transaction 4")

new\_block\_2 = {

"index": 4,

"previous\_hash": blockchain.get\_last\_block().hash,

"timestamp": int(time.time()),

"data": "Transaction 5",

"hash": blockchain.calculate\_hash(4, blockchain.get\_last\_block().hash, "Transaction 5")

}

print("\n--- Broadcasting new block from peer1 ---")

peer1.broadcast\_block(new\_block\_2)

# Add a block to peer2's chain

peer2.blockchain.add\_block("Transaction 6")

# Sync blockchain between peers

time.sleep(1)

print("\n--- Syncing blockchain from peer2 to peer1 ---")

peer2.broadcast\_block(peer2.blockchain.chain)

# Simulate peer2 trying to sync with peer1's longer chain

time.sleep(1)

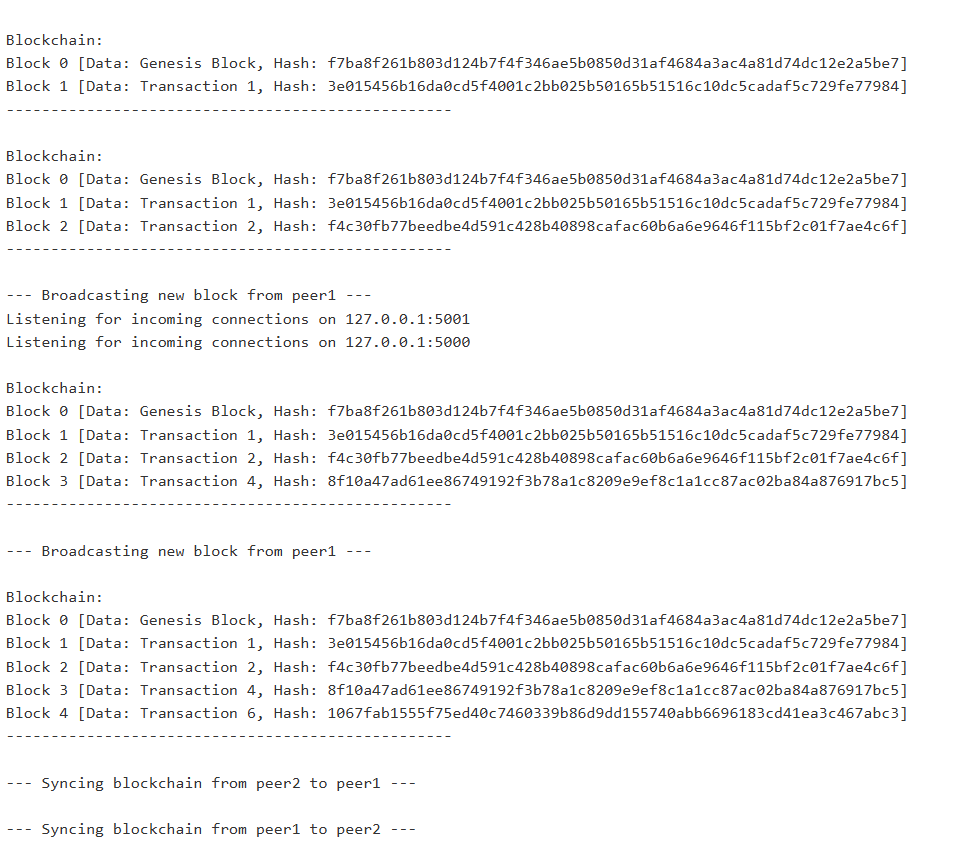
print("\n--- Syncing blockchain from peer1 to peer2 ---")

peer1.broadcast\_block(peer1.blockchain.chain)

if \_\_name\_\_ == "\_\_main\_\_":

main()

OUTPUT:



Learning Outcomes:

Understanding Blockchain Basics

Blockchain Synchronization and Integrity

Implementing Peer-to-Peer Communication