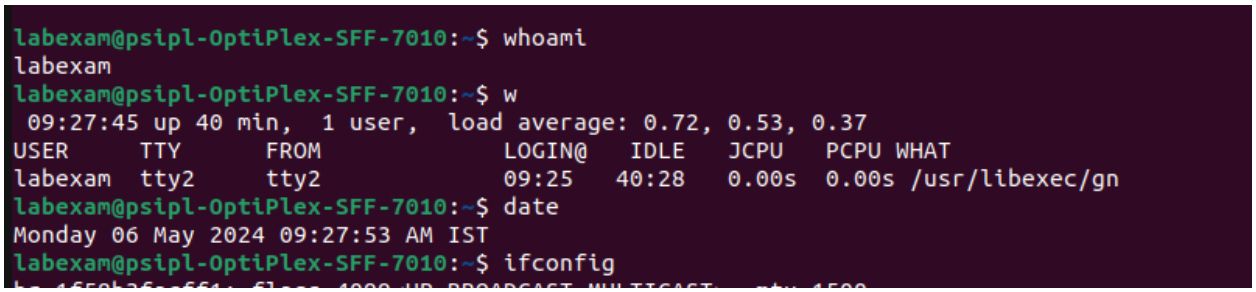
OEIT1- BCT (2023-2024)
Lab ESE

Date: 6 May 2024
Student Name: Neemisha Paul
UCID: 2021700047
Branch: C
Sem: VI

Lab title: Blockchain Programming



```
labexam@psipl-OptiPlex-SFF-7010:~$ ifconfig
br-1f58b3facff1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.19.0.1  netmask 255.255.0.0  broadcast 172.19.255.255
        ether 02:42:b7:1a:45:8e  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-ae26e2e47216: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:db:cf:90:51  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-caba9c4faf9d: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.20.0.1  netmask 255.255.0.0  broadcast 172.20.255.255
        ether 02:42:44:2f:57:5e  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:7f:3c:88:0b  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
```

Brief description of each task with screenshots (caption)
Task-1: Write a code in python to build a blockchain
Import libraries and create block

```
[6]  import hashlib
     import datetime as date
```

```
[7]  class Block:
         def __init__(self, index, timestamp, data, previous_hash):
             self.index = index
             self.timestamp = timestamp
             self.data = data
             self.previous_hash = previous_hash
             self.hash = self.calculate_hash()

         def calculate_hash(self):
             hash_string = str(self.index) + str(self.timestamp) + str(self.data) + str(self.previous_hash)
             return hashlib.sha256(hash_string.encode()).hexdigest()
```

Each block will have the following attributes:

- **Index:** This is the position of the block in the blockchain.
- **Timestamp:** This is the time at which the block was added to the chain.
- **Data:** This is the data that is stored in the block.
- **Previous Hash:** This is the cryptographic hash of the previous block in the chain.
- **Hash:** This is the cryptographic hash of the current block.

Now, we will create a Blockchain class that will define the structure of the blockchain.

```
[13] class Blockchain:
         def __init__(self):
             self.chain = [self.create_genesis_block()]

         def create_genesis_block(self):
             return Block(0, date.datetime.now(), "Genesis Block", "0")

         def get_latest_block(self):
             return self.chain[-1]

         def add_block(self, new_block):
             new_block.previous_hash = self.get_latest_block().hash
             new_block.hash = new_block.calculate_hash()
             self.chain.append(new_block)

         def is_valid(self):
             for i in range(1, len(self.chain)):
                 current_block = self.chain[i]
                 previous_block = self.chain[i-1]

                 if current_block.hash != current_block.calculate_hash():
                     return False

                 if current_block.previous_hash != previous_block.hash:
                     return False

             return True
```

Created a simple blockchain:

```
# Create the blockchain
blockchain = Blockchain()

# Add blocks to the blockchain
blockchain.add_block(Block(1, date.datetime.now(), "Transaction Data 1", ""))
blockchain.add_block(Block(2, date.datetime.now(), "Transaction Data 2", ""))
blockchain.add_block(Block(3, date.datetime.now(), "Transaction Data 3", ""))
blockchain.add_block(Block(4, date.datetime.now(), "Transaction Data 4", ""))

# Print the contents of the blockchain
for block in blockchain.chain:
    print("Block #" + str(block.index))
    print("Timestamp: " + str(block.timestamp))
    print("Data: " + block.data)
    print("Hash: " + block.hash)
    print("Previous Hash: " + block.previous_hash)
    print("\n")
```

Task-2: demonstrate the blockchain basic working

```
Block #0
Timestamp: 2024-05-06 04:46:11.842261
Data: Genesis Block
Hash: 1962e16cbf79b2317b7f7657135bd26fce3ef081eb8ee89eb82b59cc248ca150
Previous Hash: 0


Block #1
Timestamp: 2024-05-06 04:46:11.842392
Data: Transaction Data 1
Hash: 06d0d8c14fc41bf951a5f4407bf2362d3fca271c83dbfd545bf5191a39fa4ef1
Previous Hash: 1962e16cbf79b2317b7f7657135bd26fce3ef081eb8ee89eb82b59cc248ca150


Block #2
Timestamp: 2024-05-06 04:46:11.842496
Data: Transaction Data 2
Hash: c80ce7bbe92b39c074067b8c4e56439716e1d36f78f4d80a61b0ab35840e9487
Previous Hash: 06d0d8c14fc41bf951a5f4407bf2362d3fca271c83dbfd545bf5191a39fa4ef1


Block #3
Timestamp: 2024-05-06 04:46:11.843078
Data: Transaction Data 3
Hash: 581e51052eaf1387073f7fc9e2d68bfbc70db8b3855d23bce10ad2226f64f8de
Previous Hash: c80ce7bbe92b39c074067b8c4e56439716e1d36f78f4d80a61b0ab35840e9487


Block #4
Timestamp: 2024-05-06 04:46:11.843238
Data: Transaction Data 4
Hash: 9632c9daf1b09affdf50c2000c73dc8db32169cb1950ff2a773ff0c1e77154e8
Previous Hash: 581e51052eaf1387073f7fc9e2d68bfbc70db8b3855d23bce10ad2226f64f8de
```

5 blocks are getting created and each block has timestamp reflecting the time it was created data containing the transaction data for 0th it is genesis block as it is the starting block then it contains the hash of the block and the previous hash of the block to know the connection.

Task-3: Solve cryptographic puzzles for the desired prefix of the hash value of a blockchain

```python
def proof_of_work(self, previous_hash, data, leading_zeros):
    nonce = 0
    required_prefix = "0" * leading_zeros

    while True:
        block_string = f"{previous_hash}{time.time()}{data}{nonce}"
        block_hash = hashlib.sha256(block_string.encode()).hexdigest()

        if block_hash.startswith(required_prefix):
            return nonce

        nonce += 1
```

Task-4: Find Nonces for prefixes 4-zeros and 5-zeroes:

Nonce is an essential part of the proof-of-work (PoW) consensus mechanism. **The nonce, serving as a cryptographic puzzle, is a variable that miners manipulate to produce a hash value that satisfies particular requirements.**
Adding proof of work:

```python
print("Nonce for 4-zeros prefix:", blockchain.proof_of_work(blockchain.get_latest_block().hash, "New data", 4))
print("Nonce for 5-zeros prefix:", blockchain.proof_of_work(blockchain.get_latest_block().hash, "New data", 5))
```

```
Nonce for 4-zeros prefix: 621
Nonce for 5-zeros prefix: 1263398
```

History of terminal:

```
labexam@psipl-OptiPlex-SFF-7010:~/Documents$ history
    1  python labexam.py
    2  python3 labexam.py
    3  4
    4  history
labexam@psipl-OptiPlex-SFF-7010:~/Documents$ whoami
labexam
labexam@psipl-OptiPlex-SFF-7010:~/Documents$ date
Monday 06 May 2024 10:49:18 AM IST
labexam@psipl-OptiPlex-SFF-7010:~/Documents$
```