Backend / Infrastructure / SRE / Data platform Assignment

Exercise -1 : Finding the nonce.

In [1]:
```
import hashlib
```

In [4]:
```
import datetime
import hashlib
import random
import string
NONCE = 0
data = input()
n = len(data)
n
if n<=70:
  S= 100-n

d = datetime.datetime.now()
print("Start timestamp is %s",d)
ran = ''.join(random.choices(string.ascii_uppercase + string.digits, k = S))
print(str(ran))
sample = data+ran
found =0
block =123
if len(sample)==100:
    while found==0:
        z = str(NONCE)+sample
        newHash= hashlib.sha256(z.encode()).hexdigest()
        if newHash[:4]=='0000':
            found=1
        NONCE+=1
d2= datetime.datetime.now()
print("Elapsed time is %s", d2)
print(newHash)
print(NONCE)
```

```
3
Start timestamp is %s 2022-02-28 06:42:38.717000
Q02IL1D1HPP3K5SWDIFL8IIH76QMLCVUPNVT3G2KWQW47ZC1OIPLIK3NYN9GMG4H3CVBOZMO0VHOY4WBVETLHGSZ87
7KVDWJUO1
Elapsed time is %s 2022-02-28 06:42:38.837540
000033ca05d8d3533ec4b0ef5052420523a34f79acd7806bb41a5a9d74e17687
60126
```

Implementation of code:

- I kept passing block through the hashing function until I find the Nonce that gives me a hash which starts with "0000". It takes more time to find the hash that starts with more zeroes and that can be found by adding timestamp as shown below.

In [6]:
```
import datetime
import hashlib
import random
import string
NONCE = 0
data = input()
```

```
n = len(data)
n
if n<=70:
  S= 100-n

d = datetime.datetime.now()
print("Start timestamp is %s",d)
ran = ''.join(random.choices(string.ascii_uppercase + string.digits, k = S))
print(str(ran))
sample = data+ran
found =0
block =123
if len(sample)==100:
    while found==0:
        z = str(NONCE)+sample
        newHash= hashlib.sha256(z.encode()).hexdigest()
        if newHash[:6]=='000000':
            found=1
        NONCE+=1
d2= datetime.datetime.now()
print("Elapsed time is %s", d2)
print(newHash)
print(NONCE)
```

```
csdfd
Start timestamp is %s 2022-02-28 06:43:17.662864
01062BA19HIKRSX3FVF71HWBO3SN4NY1MO997WRTFXUGYO7VQEDB34UC51KWH8HIN8UDVEA6B8WN1TVUF1NTNUFNKP
3RMO8
Elapsed time is %s 2022-02-28 06:43:31.038357
000000fc7a480de51a2cd90787dbc03664150d821fbe5611f3ab7d39401a8d20
10499888
```

As we can see, the time to search increases when the number of zeroes increases

Exercise -2 : Constructing and verifying a blockchain

In [7]:

```
import datetime

import hashlib
import json
import math
from random import random



class Blockchain:


    def __init__(self):
        self.chain = []

    def create_block(self):
        block = {'nonce': len(self.chain) + 1,
                 'miner': 0,
                 }
        self.chain.append(block)
        return block

    def makenonce(self,length):
        result           = "";
        characters       = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        for i in range(length):
            result += characters[math.floor(random() * len(characters))];
```

```
        return result



    def mine_block(self,isGenesis):
        Found = False
        miner =0
        if isGenesis:
            while Found == False:
                nonce = self.makenonce(99)  ## The genesis block's nonce is 99 in length
                inputhash = str(nonce) + str(miner)
                hashval = hashlib.sha256(inputhash.encode()).hexdigest()
                if hashval[:4] == "0000":
                    Found = True
        else:
            while Found == False:
                nonce = self.makenonce(35) ##  other blocks' nonces are 35 in length.
                inputhash = str(nonce) + str(miner) + self.chain[-1]['hashval']
                hashval = hashlib.sha256(inputhash.encode()).hexdigest()
                if hashval[:4] == "0000":
                    Found = True


        response = {'nonce': nonce,
                    'miner':miner,
                    'hashval': hashval}

        self.chain.append(response)

        return response



blockchain = Blockchain()

chain = []

blockchain.mine_block(True)

for i in range(0,9):
    blockchain.mine_block(False)


#Print blockchain blocks
for i in range(0,10):
    chain.append({'nonce': blockchain.chain[i]['nonce'],
              'miner':blockchain.chain[i]['miner'],})

print(chain)


#Verify Blockchain
def verify():
    for i in range(0,10):
        if not blockchain.chain[i]['hashval'][:4] == "0000":
            print("Block invalid")
print("Blockchain verified")

verify()
```

```
[{'nonce': 'PNCNXDRKONIUIYFGMNFJHUZGCBZKAVHJDRNDJZFQXYMNQXXXFAOUBBKEFLMQDCLNJCIITBHDJDIFOW
QKNFYNPDROCVGHBMRYZBA', 'miner': 0}, {'nonce': 'JWPVJAHXWFGNGKPAJCXUWTESBVJFGWYCYNR', 'min
er': 0}, {'nonce': 'FJDDCQBYLFRZZQANSSHOSHHPUUWOHPEOOHJ', 'miner': 0}, {'nonce': 'SVVBGEFU
VHYGHIVCCYXUTWTUNCZLYPOYECB', 'miner': 0}, {'nonce': 'KYLIQSQOEKDIQUVELVLTZBWQVFKPYHQXQK
A', 'miner': 0}, {'nonce': 'BXAUAISFXUJXJCHYRYKOKPOZZUIYCUCBZDP', 'miner': 0}, {'nonce':
```

```
'GGWJSDMUELSECCHNPNPJAZFRRFMDXNUOLWC', 'miner': 0}, {'nonce': 'WZEFZXZHFEYNDCYOREYOHKFBYNB
XQCRFOCS', 'miner': 0}, {'nonce': 'EHKXPOMHYFDCAPDXKJPLFSRUCCLTUWHSFYM', 'miner': 0}, {'no
nce': 'OXOYOSTXKGNFYQHYKUKSWDKMCWCJFTOYXRA', 'miner': 0}]
Blockchain verified
```

Implementation of Code:

- A block chain is constructed by using the create_block function. The make_nonce function generates a random string which is used by the mine_block to generate the nonce in such a way that the first 4 places are filled with 0000. To achieve this padding, SHA256 is used to hash the generated nonce.

Exercise-3: Multithreading

In [8]:

```
import threading
import time
from datetime import datetime

debug_string = "Thread - {0}: [{0}] --time: {time}"
def say(i):
    dateTimeObj = datetime.now()
    timestampStr = dateTimeObj.strftime("%a %b %d %H:%M:%S %Y")
    print(debug_string.format(i, time=timestampStr))

n = int(input(("Enter value between 2 and 10[2,10)")))

thread_traker = []
if n>=2 and n<10:
    for i in range(0, n):
        thread = threading.Thread(target=say, args=(i,))
        thread.start()
        thread_traker.append(thread)
        thread.join()

# Threads will automatically get stopped, but if you want to stop one of it specifically
# if you want to kill all run it in a loop
kill = int(input())
stop_threads = True
thread_traker[kill - 1].join()
# or use sys.exit() to kill all other threads run by programs
```

```
Enter value between 2 and 10[2,10)5
Thread - 0: [0] --time: Mon Feb 28 06:44:02 2022
Thread - 1: [1] --time: Mon Feb 28 06:44:02 2022
Thread - 2: [2] --time: Mon Feb 28 06:44:02 2022
Thread - 3: [3] --time: Mon Feb 28 06:44:02 2022
Thread - 4: [4] --time: Mon Feb 28 06:44:02 2022
4
```

Implementation of Code:

- A thread-traker list is created to append all the threads to the list. N is the number of threads we want the program to implement. If condition is given such that the n value lies between 2 and 10. The thread calls the say method which prints the required output

Exercise- 4 : Inter-thread communication

In [9]:

```
from queue import Queue
import threading
import random
import string
```

```
import time

def produce_and_consume(i,hmap):

    #Produce the data and push it to the queues of all threads
    length =10
    letters = string.ascii_uppercase
    data = ''.join(random.choice(letters) for i in range(length))
    # out_q.put(data)
    for k in hmap.keys():
        if k!=i:
            hmap[k].put(data)
    print("Thread {0} sending message {1} to all other threads\n".format(i, data) )

    time.sleep(5)

    #consume the data
    while True:
        while hmap[i].qsize() > 0:
            print("Thread{0} received a message {1}\n".format(i,hmap[i].get()))
        time.sleep(5)

hmap = {}
n = int(input())
if n>=2 and n<6:
    for i in range(0,n):
        hmap[i] = Queue()

    for i in range(0, n):
        thread = threading.Thread(target=produce_and_consume, args=(i,hmap))
        thread.start()
```

```
5
Thread 0 sending message PQWRAVEZVM to all other threads
Thread 1 sending message MEMWFQZOSG to all other threads


Thread 2 sending message UTHFRYOYKC to all other threads

Thread 3 sending message SSHUWLJSQL to all other threads

Thread 4 sending message QMFSUUQSIL to all other threads
```

Implementation of Code:

- A hash map makes use of a hash function to compute an index with a key into an array of buckets or slots. Its value is mapped to the bucket with the corresponding index. Hmap is used to send the threads accordingly to consume and produce methods. The produce and consume method is used to send the messages from one thread thread and receive messages by all other threads.

Exercise - 5: Decentralizing the blockchain

In [72]:

```
import datetime

import hashlib
import json
import math
from random import random
from queue import Queue
import threading
import random
```

```
import string
import time



class Blockchain:


    def init(self):
        self.chain = []

    def create_block(self):
        block = {'nonce': len(self.chain) + 1,
                 'miner': 0,
                 }
        self.chain.append(block)
        return block

    def makenonce(self,length):
        result           = "";
        characters       = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        for i in range(length):
            result += characters[int(math.floor(random.random() * len(characters)))];

        return result



    def mine_block(self,isGenesis,chain,i):
        Found = False

        miner = i
        if isGenesis:
            while Found == False:
                nonce = self.makenonce(99)
                inputhash = str(nonce) + str(miner)
                hashval = hashlib.sha256(inputhash.encode()).hexdigest()
                if hashval[:4] == "0000":
                    Found = True
        else:
            while Found == False:
                nonce = self.makenonce(35)
                inputhash = str(nonce) + str(miner) + self.chain[-1]['hashval']
                hashval = hashlib.sha256(inputhash.encode()).hexdigest()
                if hashval[:4] == "0000":
                    Found = True


        response = {'nonce': nonce,
                    'miner':miner,
                   'hasval':hashval}

        print (response)
        chain.append(response)


        return response




# #Verify Blockchain
# def verify():
#     for i in range(0,10):
#         if not blockchain.chain[i]['hashval'][:4] == "0000":
#             print("Block invalid")
```

```
#Verify Blockchain
def verify(chain):
    for i in range(0,10):
        if not chain[i]['hashval'][:4] == "0000":
            print("Block invalid")
    print("Blockchain is verified")


def produce_and_consume(blockchain,chain,hmap,i):
    #Produce the data and push it to the queues of all threads

    response = blockchain.mine_block(True,chain,i)

    for k in hmap.keys():
        if k!=i:
            chain.append(response);
            hmap[k].put(chain)

    time.sleep(5)

    #consume the data
    while True:
        while hmap[i].qsize() > 0:
            incomingChain =  hmap[i]
            if verify(incomingChain) and len(incomingChain) > len(chain):
                chain = incomingChain


        time.sleep(5)



blockchain = Blockchain()
chain = []
hmap = {}
n = int(input())
for i in range(0,n):
    hmap[i] = Queue()


for i in range(0,n):
    thread = threading.Thread(target=produce_and_consume, args=(blockchain,chain,hmap,i))
    thread.start()

print(chain)
# while (len(chain) <= 10):
#     newBlock = blockchain.mine_block(True)
#     if newBlock is not None:
#         chain.append(newBlock)
#         #send chain to all other threads
#     thread = threading.Thread(target=blockchain.mine_block(True), args=(chain,hmap))
#     thread.start()
```

```
3
[]
{'nonce': 'FSPKLFESLKKYWIBMFFNHBAXEWLWVECUWFWSMEJTZTLGKPQUWLWXIVRQQHLNINDMOJSOTEUEACSQUTJV
VMBHDMDOXGYGJFIBSKZA', 'miner': 2, 'hasval': '0000362f55f1063301aba9542ead143c7a2d064e0ee8
1f3e6dddae7f89b48ad8'}
{'nonce': 'ILPTNRNKJRUOSUQUDTNKDHUJVBSEPHQHWVJYKWKSIYSSQHCCXAFHHOPQFWTVBWFPIBEBEBZESVUZJWF
HCQLEZGAUNHQCSNJZVFK', 'miner': 0, 'hasval': '00002074cbe9038474206aba225d6f334d0516375344
9ba828585dc1cf07b7c6'}
{'nonce': 'RZRPDAQECXYKTZWLZHYRNAJEDWQEAPDJGITPLRWWEEKINNNUPJAGSMHCGQBELCLGRADGEOJFSZDYUAV
```

```
MDBNIJAZZAXLMDTABURG', 'miner': 1, 'hasval': '00002809560d2cb672c93286b963a2057537201cf35a
d42da68d8ae08918f96d'}
Exception in thread Thread-204:
Traceback (most recent call last):
  File "/Users/anitateladevalapalli/opt/anaconda3/lib/python3.9/threading.py", line 973, i
n _bootstrap_inner
    self.run()
  File "/Users/anitateladevalapalli/opt/anaconda3/lib/python3.9/threading.py", line 910, i
n run
    self._target(*self._args, **self._kwargs)
  File "/var/folders/6f/c2b7vdpx247cstzj573kd1k40000gn/T/ipykernel_71572/1254715824.py", l
ine 103, in produce_and_consume
  File "/var/folders/6f/c2b7vdpx247cstzj573kd1k40000gn/T/ipykernel_71572/1254715824.py", l
ine 82, in verify
TypeError: 'Queue' object is not subscriptable
Exception in thread Thread-202:
Traceback (most recent call last):
  File "/Users/anitateladevalapalli/opt/anaconda3/lib/python3.9/threading.py", line 973, i
n _bootstrap_inner
    self.run()
  File "/Users/anitateladevalapalli/opt/anaconda3/lib/python3.9/threading.py", line 910, i
n run
    self._target(*self._args, **self._kwargs)
  File "/var/folders/6f/c2b7vdpx247cstzj573kd1k40000gn/T/ipykernel_71572/1254715824.py", l
ine 103, in produce_and_consume
  File "/var/folders/6f/c2b7vdpx247cstzj573kd1k40000gn/T/ipykernel_71572/1254715824.py", l
ine 82, in verify
TypeError: 'Queue' object is not subscriptable
Exception in thread Thread-203:
Traceback (most recent call last):
  File "/Users/anitateladevalapalli/opt/anaconda3/lib/python3.9/threading.py", line 973, i
n _bootstrap_inner
    self.run()
  File "/Users/anitateladevalapalli/opt/anaconda3/lib/python3.9/threading.py", line 910, i
n run
    self._target(*self._args, **self._kwargs)
  File "/var/folders/6f/c2b7vdpx247cstzj573kd1k40000gn/T/ipykernel_71572/1254715824.py", l
ine 103, in produce_and_consume
  File "/var/folders/6f/c2b7vdpx247cstzj573kd1k40000gn/T/ipykernel_71572/1254715824.py", l
ine 82, in verify
TypeError: 'Queue' object is not subscriptable
```

Implementation of code:

- Started n threads initially and each of the thread mines the block in the producer function and in consumer each of the thread gets the updated chain and the verify function is called.