Birla Institute of Technology & Science, Pilani
Hyderabad Campus

Blockchain Assignment - 2

Student ID	*Name*
Anjali M	*2020H1030116H*
Swati Sharma	*2020H1030149H*
Divyarth Singh	*2020H1030113H*

Table of Contents

1. Introduction:

In the previous submission, we implemented a basic blockchain that employed PoW (Proof of Work consensus), stored and updated the transactions in a database file. Continuing with the same blockchain architecture, as a part of the second task, we've implemented PoET consensus algorithm.

Supported by Intel Corporation 2017 equipped SGX-Software Guard Extensions, PoET is a permissioned consensus algorithm where a random time to sleep is chosen for each node and the first node to wake up gets added to the blockchain. Culling the need for hashing power that was needed to provide a truly random assignment that was needed in PoW, PoET decreases the power and resource consumption by miners.

Every network participating node must wait for the randomly selected amount of time, and the new block wins the first node to finish the assigned waiting time. Each blockchain node creates a random time to wait and sleep for the length given. The one with lowest wait time wakes up first, sending the blockchain a new block to the whole peer network and transmitting information. The identical procedure is repeated for newer blocks.

2. Changes to the previous code and Libraries:

We added new logic so that Dexter would be the only receiving user in any transaction. We also limited the blocksize to a maximum of 3 transactions.

Following libraries were used:
random- for using random library to assign random time values for each node with in a given specified range
ecdsa (Elliptic Curve Digital Signature Algorithm) - to create public and private keys for validating the new nodes
sqlite3- for storing user details and modifying according to transactions
time- for generating timestamps and random time and sleep function

3. Implementing PoET

In simple terms, logic used to implement PoET is as follows:

- A node which wants to join the verified nodes has to be validated and it is done by creating private and public keys using the ECDSA library.

- The node forwards this key when requesting to join the network. The nodes that are already a part of the network verify this key.

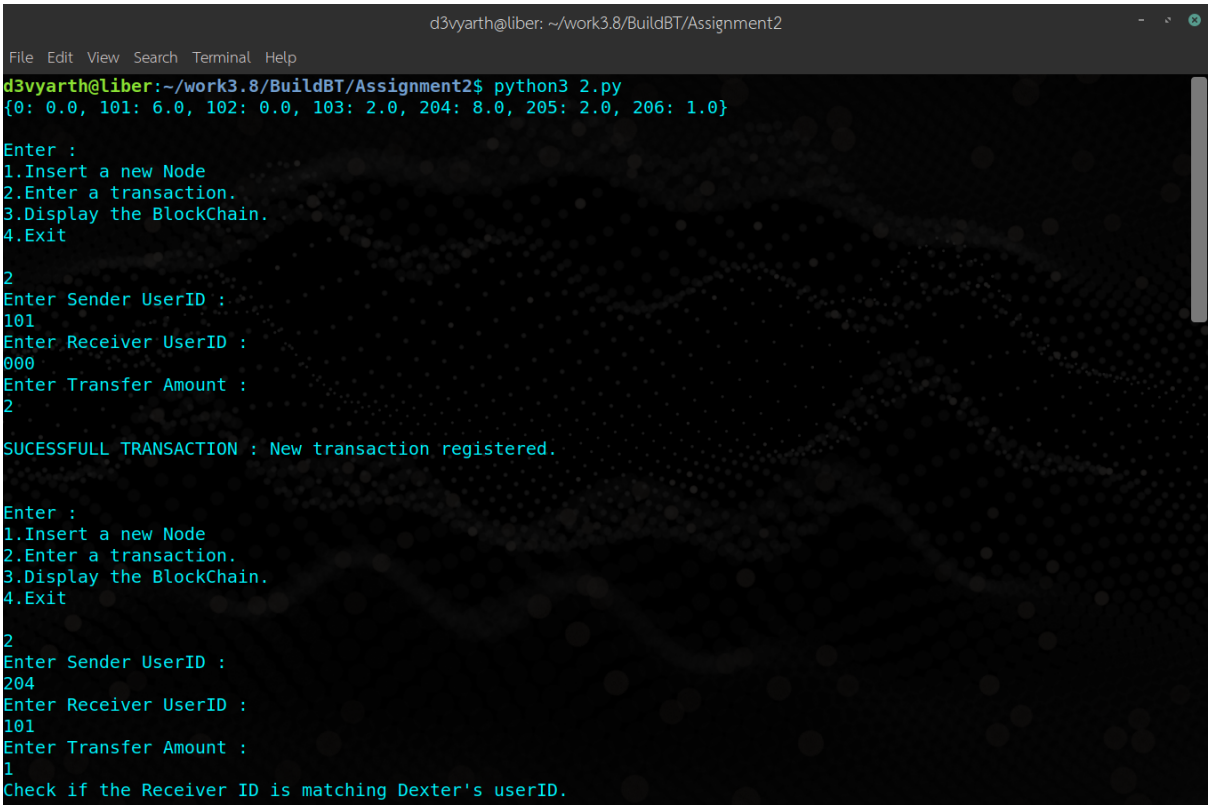
- To select the leader of the nodes, or the node which creates the block

which is linked to the existing blockchain, PoET algorithm initializes all the with a random time; the first one whose time expires becomes the winner. This means that it creates a new block. (Fairness of Algorithm is based on the randomness of the timers given to the nodes in this process.)

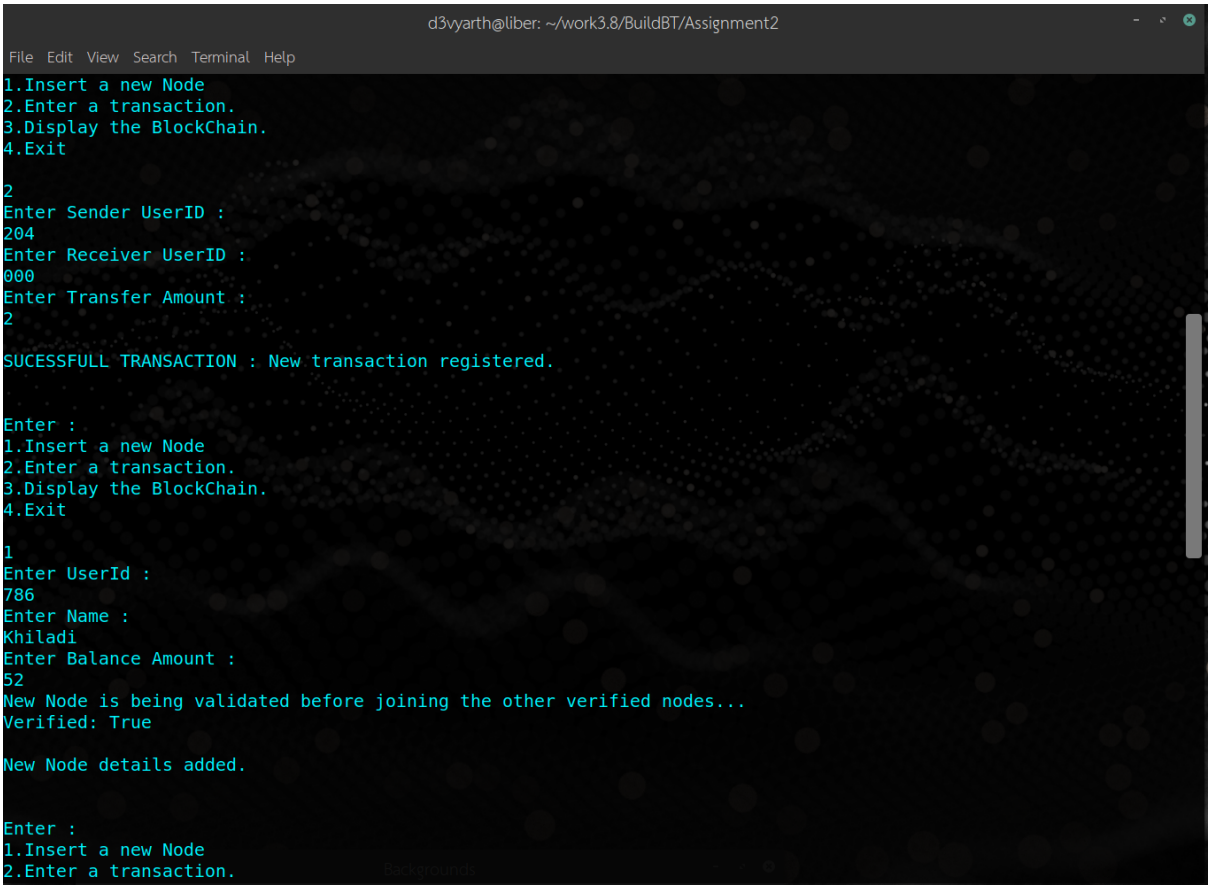
- Later this news is broadcasted to the remaining nodes. (In our implementation, to indicate the receiving of this news, we have reset the random time values of all the nodes to zero.)

4. Code Execution Snapshots

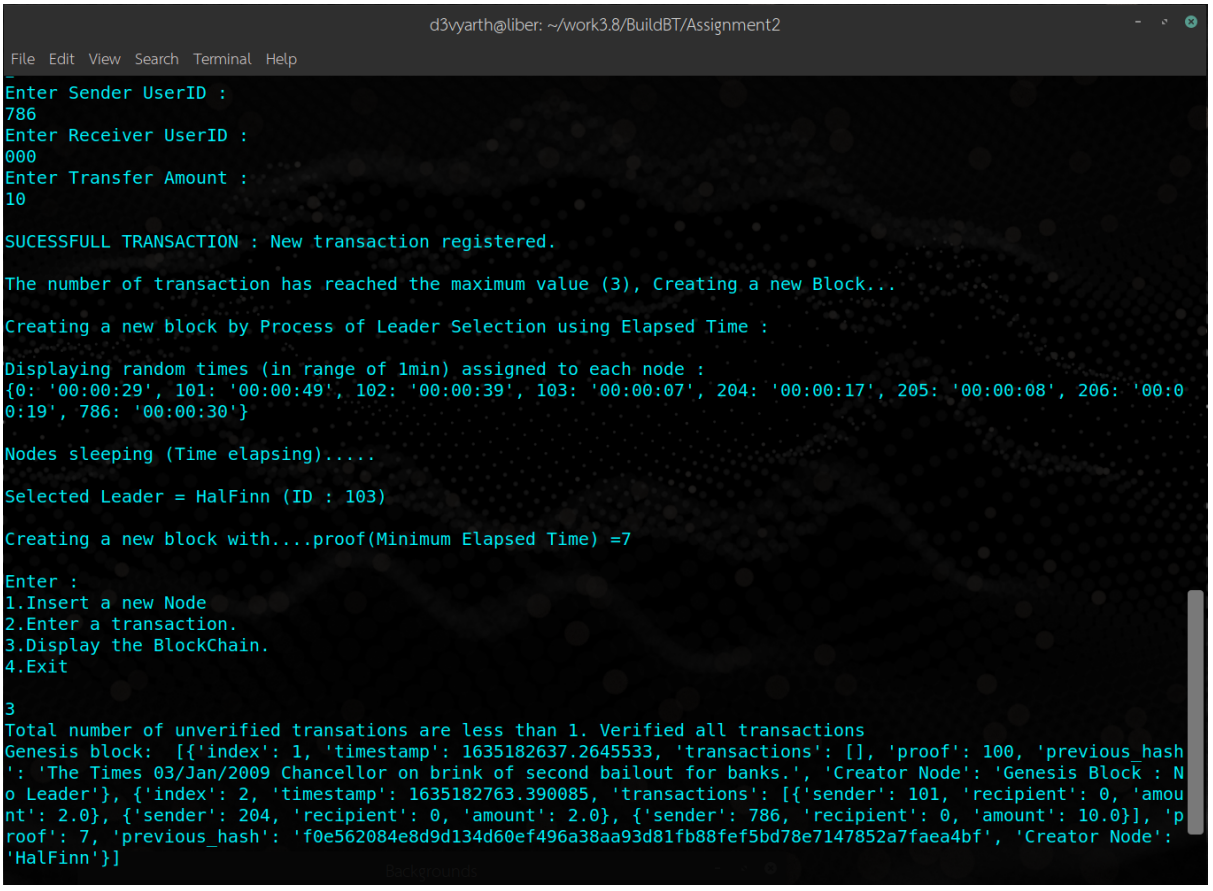
All possible corner cases have been covered. The output is being stored in Userdb.db as well as getting displayed on the console. The console also displays the time elapsed and leader selection.



(Carrying out transaction to Dexter and a corner case when receiver is not Dexter)



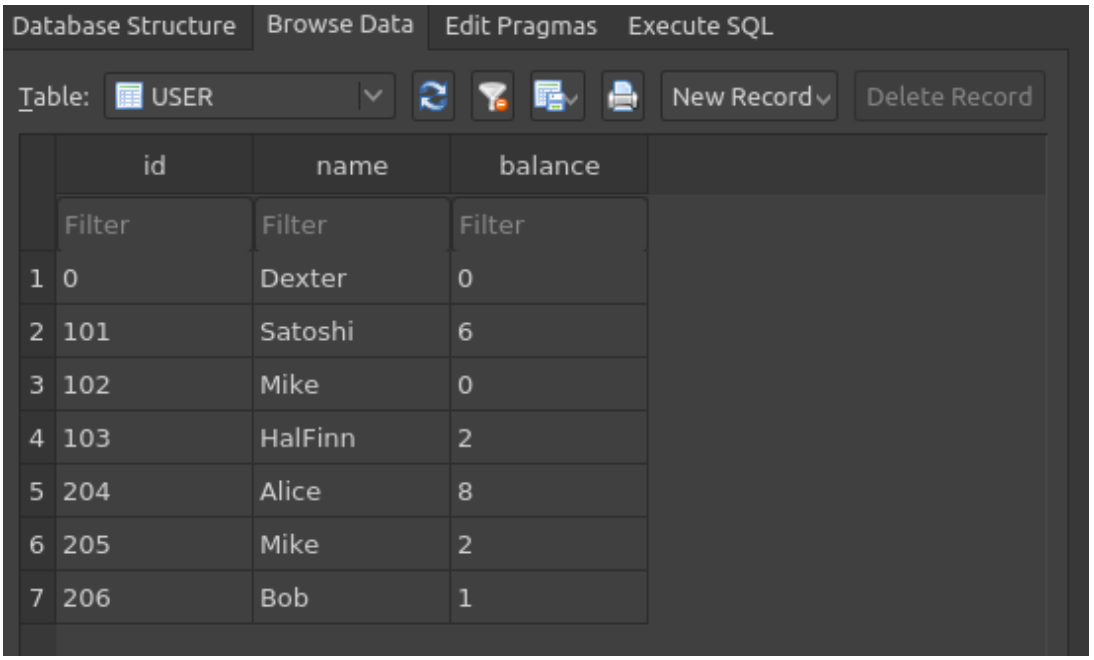
(Node Validation and transaction)

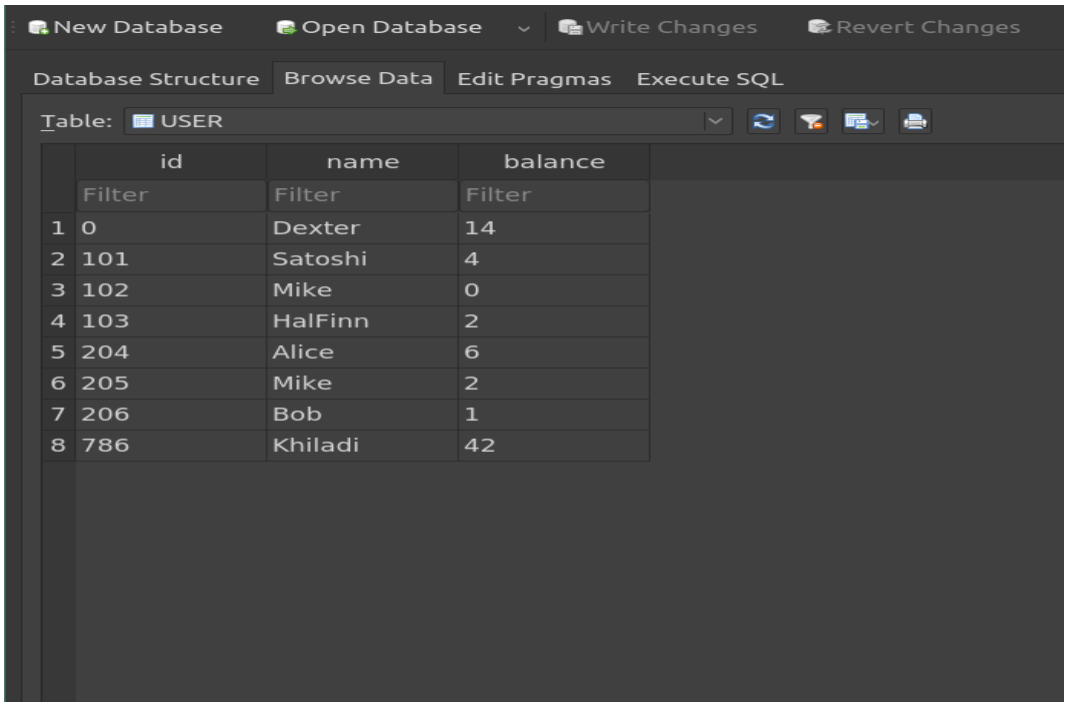


(Displaying the Blockchain)

```
The number of transaction has reached the maximum value (3), Creating a new Block...

Creating a new block by Process of Leader Selection using Elapsed Time :

Displaying random times (in range of 1min) assigned to each node :
{0: '00:00:03', 101: '00:00:49', 102: '00:00:50', 103: '00:00:14', 204: '00:00:27', 205: '00:00:59', 206: '00:00:54
'}

Nodes sleeping (Time elapsing).....

Selected Leader = Dexter (ID : 0)

Creating a new block with....proof(Minimum Elapsed Time) =3

After the news of the elected leader is broadcasted to all the remaining nodes their random time values are reset
{0: 0, 101: 0, 102: 0, 103: 0, 204: 0, 205: 0, 206: 0}

Enter :
1.Insert a new Node
2.Enter a transaction.
3.Display the BlockChain.
4.Exit
```

(Broadcasting the elected leader to all the nodes)





(Initial and Modified User DB)