**Exercise 2 – MtaCoin**

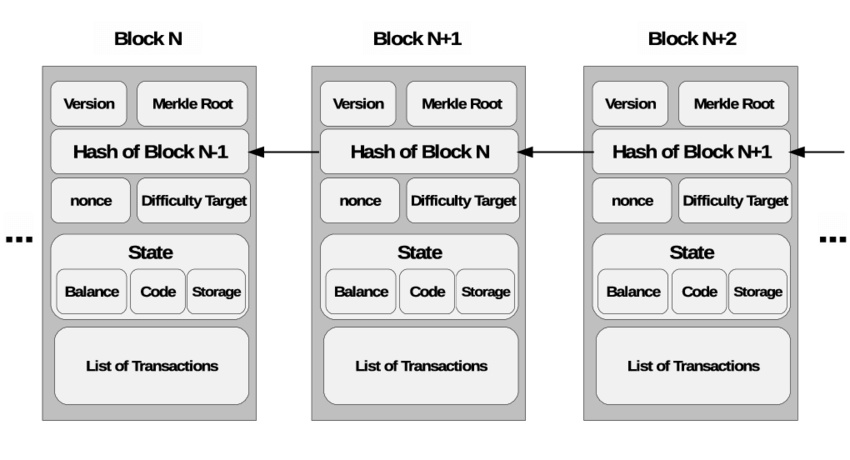
**Objective:**

Practice synchronization between consumer and producer threads in the same process.

Execute multithreads application that illustrates a cryptocurrency blockchain network in which there are 4 mining threads that mine blocks (calculate hashes) and send these blocks to a server that verifies and adds these blocks to its blockchain.

**Blockchain Overview**

A growing list of records, called blocks, that are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. (<https://en.wikipedia.org/wiki/Blockchain>)



**Cryptographic hash**

An algorithm that can be run on data (a block in our case) and output an irreversible and unique(almost) checksum of it. Usually SHA256 is used but we will use a simple checksum function for now, **crc32**(part of zlib library).

**crc32:** <https://en.wikipedia.org/wiki/Cyclic_redundancy_check>

**Mining**

The act of calculating a hash(checksum) on a given block, once found appending it to the end of the blockchain and populating it to the other miners.

**Proof of Work**

A proof of work is a piece of data which is difficult (costly, time-consuming) to produce but easy for others to verify and which satisfies certain requirements. Producing a proof of work can be a random process with low probability so that a lot of trial and error is required on average before a valid proof of work is generated. Bitcoin achieves this by making the miners hash many combinations of letters and numbers until the resulting hash contains a specific number of leading “0”s. In our case, because we use crc32, we will check for leading 0 bits.

**Assignment description:**

Write a C program that is divided into 4 miner threads and 1 server thread:

**Block structure**

typedef struct {

int height; // Incrementeal ID of the block in the chain

int timestamp; // Time of the mine in seconds since epoch

unsigned int hash; // Current block hash value

unsigned int prev\_hash; // Hash value of the previous block

int difficulty; // Amount of preceding zeros in the hash

int nonce; // Incremental integer to change the hash value

int relayed\_by; // Miner ID

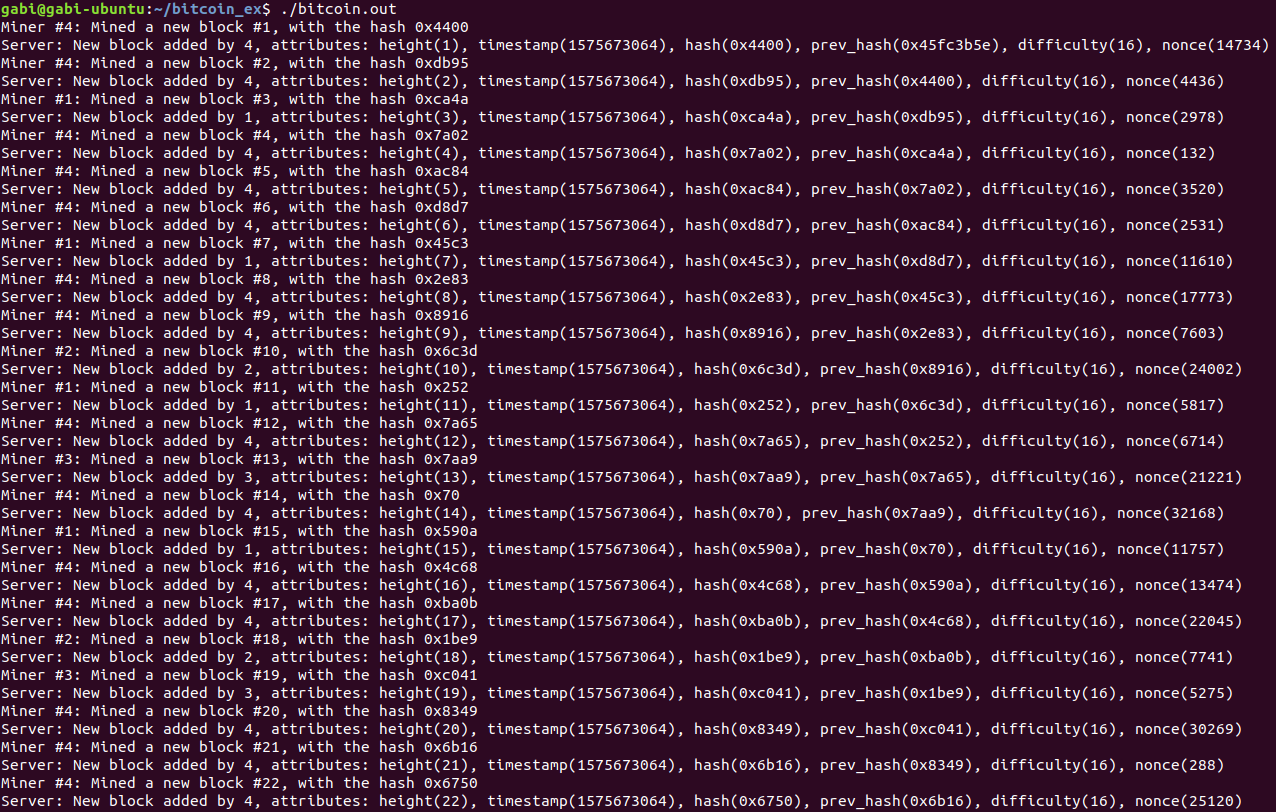
} BLOCK\_T;

**Server thread**

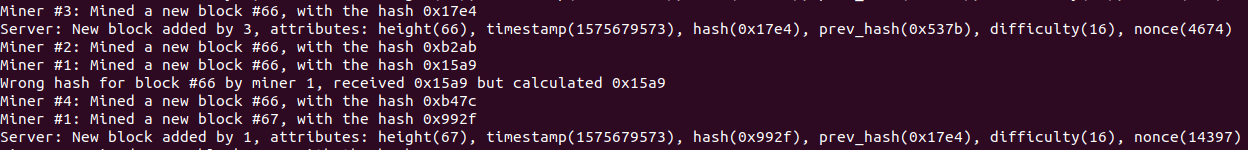
1. Generate an initial genesis block
2. Notify the miners they have a new block
3. Wait for notification from one of the miners about a successfully mined block
4. Verify its proof-of-work (run same hash function)
5. If proved, append this block to the head of the chain and notify all miners there is a new block (place height, prev\_hash and difficulty in the new block)
6. If not proved\*, print error and leave the chain as is
7. Go to 3

**Miner threads**

1. Start from the current block the server point to, this block should already hold, height prev\_hash and difficulty), set up by the server, miner shall add it’s miner\_id in the relayed\_by field
2. If new block was appended the head of the chain, go to 1
3. nonce++
4. Update timestamp in the block with current time
5. call crc32 function on the following fields on the block, **height**, **timestamp**, **prev\_hash**, **nonce**, **relayed\_by**
6. Upon successful work, crc32 result has same amount of zero bits or more then specified in difficulty field, notify the server with new the block data



\* Make sure to handle failure scenarios such as mismatch in the hash value, block height out of order and other errors you might encounter, for example:



\* Add additional miner thread that send an invalid block to the server every 1 second(sleeps rest of the time), the purpose of it is to test the server drops invalid blocks.