Simulating Proof of useful work in C/C++

Objective

Here we demonstrate how a custom problem can be solved on the blockchain.
We use threads in C to simulate a miner.

Files

project.cpp

It is the main program. It does the mining.
Blockchain is stored in outputs/blockchain.txt
The useful work extracted from mining is stored in outputs/useful_information.txt

compile

```
g++ project.cpp -lpthread -lcrypto
```

run

```
./a.out 20 10
```

1st argument = number of miners

2nd argument = number of blocks to mine

verify.cpp

It is an auxillary program.
It verifies if the blockchain *(stored in outputs/blockchain.txt)* is valid.

compile

```
g++ verify.cpp -lcrypto
```

run

```
./a.out
```

Directory Structure:

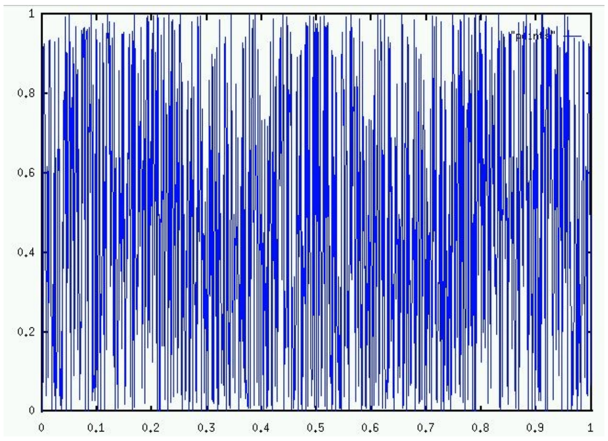
```
project
│   a.out
│   README.pdf
│   project.cpp
│   verify.cpp
│
└── helper
│   └── hash.h
│
└── outputs
    ├── blockchain.txt
    └── useful_information.txt
```

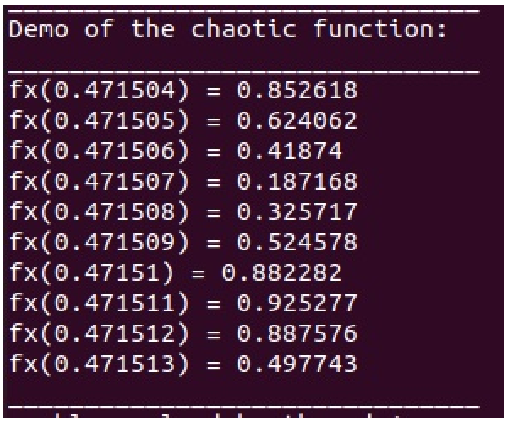
Custom Problem

It is a chaotic function: **implementation**

Problem is to try and find x such that f(x) < y_0 and x is an integral multiple of 2^{-20}

We can see below-right that incrementing by 10^{-6} produces very different outputs and $10^{-6} \approx 2^{-20}$
hence if we want to solve the above mentioned problem, there is no better way than to randomly guess values.
The more we mine, the more information we will get about this chaotic function,and we can use these results wherever required
we can also swap in similar problems and extract useful information while mining
Hence our blockchain solution is meeting the requirement (It does useful work which can be used by others)





```
-----------------------------------
Demo of the chaotic function:

-----------------------------------
fx(0.471504) = 0.852618
fx(0.471505) = 0.624062
fx(0.471506) = 0.41874
fx(0.471507) = 0.187168
fx(0.471508) = 0.325717
fx(0.471509) = 0.524578
fx(0.47151) = 0.882282
fx(0.471511) = 0.925277
fx(0.471512) = 0.887576
fx(0.471513) = 0.497743

-----------------------------------
```

Approximations and Assumptions

- We are dealing with very few nodes (<1000)
- The likelihood of any node not receiving broadcast message is ≈ 0
- Hence we can assume all the nodes will have identical blockchains
- Therefore it is sufficient to maintain a single copy of the blockchain
- Blocks are added to this must be verified by >= $\lceil n/2 \rceil$ nodes
- miners implemented with pthreads
- broadcast messages and state of the blockchain implemented with global variables (memory is shared by all threads/miners)
- for every block, the objective value y_0 is randomly determined