DESCRIPTION:

The goal of this first project is to use Erlang and the Actor Model to build a good solution to this problem that runs well on multi-core machines. We may divide the burden and employ several cores for speedier processing by building an actor-model framework since Bitcoin mining using SHA256 hashing is a time-consuming and intensive computational operation.

INSTRUCTIONS TO RUN CODE:

- There are 6 files in this project. Of these files, 2 files will have to be compiled as follows:

  c(main).
  c(test).

- Open 2 terminals, and after navigating to the source folder execute the following commands:

  Terminal 1:
  erl -sname anil(ip address of system 1)

  Terminal 2:
  erl -sname likhitha(ip address of system 2)

- After initializing the boss, server with above commands, run the following commands:

  Terminal 1:
  main.start(4). [4 corresponds to the number of leading 0's. Can be changed accordingly.]

  Terminal 2:
  test.start('anil@ipaddress'). [Worker nodes are initialized on a remote server.

OBSERVATION :

1. With 16 worker units spawned, we are able to use all 8 of my computer's cores; if I increase or reduce this number, my Real to CPU time ratio falls. For various needed leading zero counts, this behavior changes. We arrived at this figure using a hit-and-miss methodology.

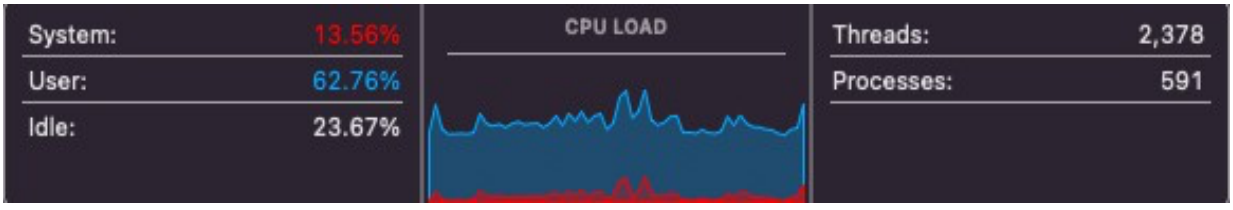
2. The result for running the code with 4 leading zeros is as follows:

```
(anil@10.20.48.15)1> master:start(4).
Server Side :
Input String :  kondapalliqkUBafHdJE5Y
SHA256 HashCode :  000008423ea344ba32710d6d18fe18692b5304d286e4c296eba8180942239ee8
Server Side :
Input String :  kondapallitGeoE890Wuz4
SHA256 HashCode :  00002a120a87fded7b16be3027405e758dd8963a508f2441b938f38b3f1a8f07
Server Side :
Input String :  kondapalliA9pa+HIGCIcX
SHA256 HashCode :  000005ed383f4575ab555b9f69021211dcdeddb6bbce5ec4820d343fd54a4cf8
Server Side :
Input String :  kondapalliqs6blSVBQGIF
SHA256 HashCode :  00004034e9ede028611f0147a0b45a0e0e3c7b00688cb3915ef69520482e267e
Server Side :
Input String :  kondapalliqQ8LulxEPB7b
SHA256 HashCode :  000079568f0db2358ee0bdf935bca827e138db08d29d6403286144f4a9401ef6
```

3. Real time VS CPU time comparison with different leading 0 and actors:

Leading Zeroes	Actors Count	Real time	CPU time	Ratio
4	8	00:06.563	00:13.148	2.003
	16	00:01.778	00:08.462	4.759
	32	00:04.929	00:25.413	5.155

With leading zeros = 4 and 8 cores, the real-time and CPU time ratio turns out to be **2.003.** This shows that our model is implementing the parallelism concept. With increase in actors for the same leading zeros hash, the ratio also increases gradually whilst utilizing the cores perfectly.



4. The coin with the most leading 0s we managed to find was **7**. Although, we were also able to compute the coins with **6** leading zeros faster as compared to **7**.

```
(anil@10.20.48.15)1> main:start(7).
Server Side :
Input String :  kondapalliGULMFHIIf2tn
SHA256 HashCode :  00000004d034e071f22bb18eb76eb9387e795e873e16aaab785d04c84858b339
```

```
(anil@10.20.48.15)1> master:start(6).
Server Side :
Input String :  kondapalliLukkCqZE6U7m
SHA256 HashCode :  00000043524a6b52978719792d7b41ae888c853567d323f707a0b9cc013ea248
Server Side :
Input String :  kondapalligN5bR3ZOYqtD
SHA256 HashCode :  000000dddb5ddb372cc578a627ba0cea94fa237f839fec7806c8958f4e8ddc46
Server Side :
Input String :  kondapalliG2bI+8KxecaC
SHA256 HashCode :  000000b672112948bc0fe6aaecd07c705e518b47150d592788cefdbd5390de18
Server Side :
Input String :  kondapalliQgGOdTIcfWq8
SHA256 HashCode :  00000099797ea30daf0f86d4e60f40cbfe3d85780763bebed4488097e1379a1a
Server Side :
Input String :  kondapallid9pOPpCSHNlD
SHA256 HashCode :  000000a32d115aa9f66a4aed8b82b0184f212a5fddcd22db4bea4ec69576bdf1
Server Side :
Input String :  kondapalliCEKk1y/atTwf
SHA256 HashCode :  000000644a4d42cafc9734059a88588df2165a8ca11dc727e17849b39b4a9e0c
```

5. We were able to run the code on 5 working machines simultaneously and successfully.

   Running with 5 machines
   24056.893u 173.411s 1:41:33.35 397.6%
   1,870 coins found.

   The performance increased significantly from 1 to 5 machines. Adding more machines gave performance increases with diminishing returns. The main program would need to be able to scale properly with the larger number of workers.