COP5615- Distributed Operating System Principles
Project 1: Bitcoin Mining

Rishitha Reddy Kallu
UFID: 20015890

Abhijith Chandra Mergu
UFID: 33991217

Input:

The input to the program is an integer value "n" which specifies how many leading zeros must be present in the hashed value of the string for the program to find a string value that will satisfy this constraint. The purpose of this project is to create a solid, multi-core-compatible solution to this problem using Erlang and the Actor Model.

Output:

The output would be the list of strings which satisfied the requirement and their corresponding hashes.

Implementation:

The program involves six major components – server, client, worker, logic, spawner, timer.

- **Server** – The supervisor process which calls **spawner** based on the received messages, it is written generic to handle actor spawns on both local server machine and the client machines.
- **Worker** – The individual actor which executes the logic to create, hash and then verify strings.
- **Client** – The entry point for the client machines, which registers a client to the server.
- **Logic** – The core logic function to hash and verify the leading zero presence.
- **Spawner** – The wrapper function which spawns actors on the given node. It is written generic so that it can spawn actors on both server and client machines.
- **Timer** – The timer process calculates and prints the real time and CPU time spent by the program when the server is shutdown.

The workflow for the program goes something like this:

1. When the server:start() is run, the program initiates a timer and then creates a server actor by calling spawn. And then sends server process a message **selfSolve**, this basically translates to spawn the actors on the same machine.
2. Now, the server process receives the **selfSolve** and then spawns the workers on the current node. The workers are spawned by calling a wrapper **Spawner** which creates the actors on the same node. And then moves on to wait for the other messages.
3. The workers which are spawned on the current machine keeps on running the **logic** function iteratively to find the suitable strings. The logic function has the core logic which finds a

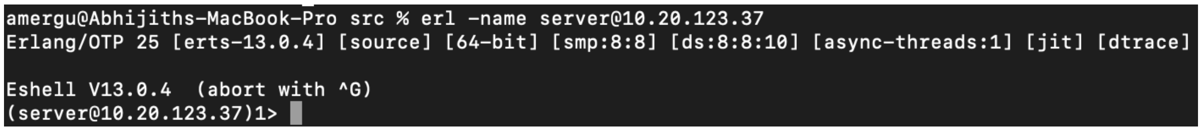
random string by taking a random length and the string content is formed by picking characters randomly from the charset passed.

4. The suitable strings which are found are printed by the spawner function.
5. Now comes the client, on the client side when **server:client(<ip address of server>)** is run, the client finds the server process PID by making a rpc call and then sends a message to register itself along with its node.
6. Once the server receives the register message it calls **spawner** on the received **client node** and thus the workers are spawned on the client machine and the results are printed on the server machine.

How to run:

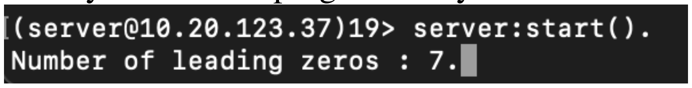
Run Server:

1. Initialize the erlang shell using the **erl** command with a **-name** parameter as **server** appended with the ip of the machine at the location where your erlang source files are located.



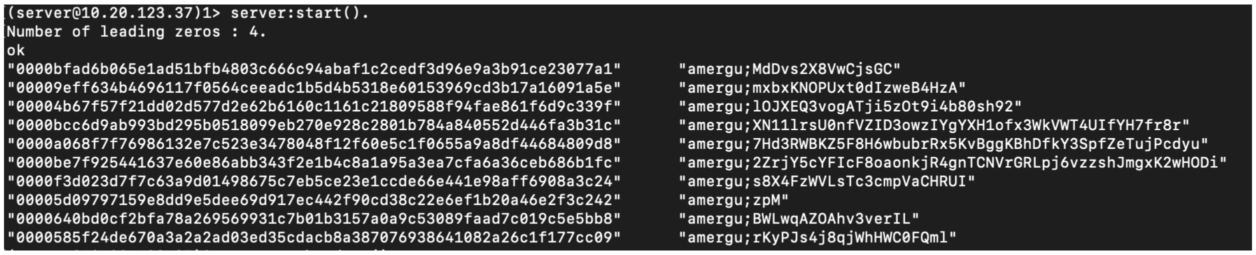
```
amergu@Abhijiths-MacBook-Pro src % erl -name server@10.20.123.37
Erlang/OTP 25 [erts-13.0.4] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [jit] [dtrace]

Eshell V13.0.4  (abort with ^G)
(server@10.20.123.37)1> 
```

2. Start the server by running **server:start()** and then enter the number of leading zeros you want the program to try.



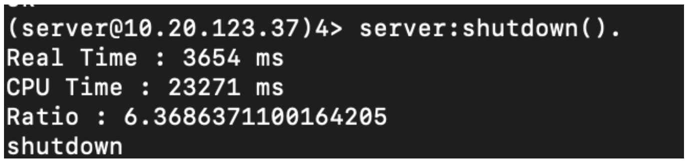
```
(server@10.20.123.37)19> server:start().
Number of leading zeros : 7.
```

3. At this point the workers are already spawned and working on the problem. And server is listening for any new clients that try to register. You can go ahead to the client machine and start running client commands. You can see a list of strings and their corresponding hashes that are being printed.



```
(server@10.20.123.37)1> server:start().
Number of leading zeros : 4.
ok
"0000bfad6b065e1ad51bfb4803c666c94abaf1c2cedf3d96e9a3b91ce23077a1"      "amergu;MdDvs2X8VwCjsGC"
"00009eff634b4696117f0564ceeadc1b5d4b5318e60153969cd3b17a16091a5e"      "amergu;mxbxKNOPUxt0dIzweB4HzA"
"00004b67f57f21dd02d577d2e62b6160c1161c21809588f94fae861f6d9c339f"      "amergu;lOJXEQ3vogATji5zOt9i4b80sh92"
"0000bcc6d9ab993bd295b0518099eb270e928c2801b784a840552d446fa3b31c"      "amergu;XN11lrsU0nfVZID3owzIYgYXH1ofx3WkVWT4UIfYH7fr8r"
"0000a068f7f76986132e7c523e3478048f12f60e5c1f0655a9a8df44684809d8"      "amergu;7Hd3RWBKZ5F8H6wbubrRx5KvBggKBhDfkY3SpfZeTujPcdyu"
"0000be7f925441637e60e86abb343f2e1b4c8a1a95a3ea7cfa6a36ceb686b1fc"      "amergu;2ZrjY5cYFIcF8oaonkjR4gnTCNVrGRLpj6vzzshJmgxK2wHODi"
"0000f3d023d7f7c63a9d01498675c7eb5ce23e1ccde66e441e98aff6908a3c24"      "amergu;s8X4FzWVLsTc3cmpVaCHRUI"
"00005d09797159e8dd9e5dee69d917ec442f90cd38c22e6ef1b20a46e2f3c242"      "amergu;zpM"
"0000640bd0cf2bfa78a269569931c7b01b3157a0a9c53089faad7c019c5e5bb8"      "amergu;BWLwqAZOAhv3verIL"
"0000585f24de670a3a2a2ad03ed35cdacb8a387076938641082a26c1f177cc09"      "amergu;rKyPJs4j8qjWhHWC0FQml"
```

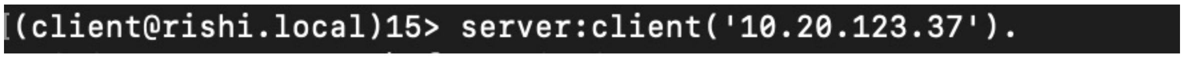
4. Once you've enough strings you can run the **server:shutdown()**. This ends the server process and prints the time related statistics. And your server would no longer serve any clients.



```
(server@10.20.123.37)4> server:shutdown().
Real Time : 3654 ms
CPU Time : 23271 ms
Ratio : 6.3686371100164205
shutdown
```

Run Client:

1. Initialize the erlang shell using the **erl** command at the location where your erlang client source file is located.
2. Start the client by running **server:client(<ip address of the server>)**. Give the ip of the server machine as a parameter, this will register the client to the server. And then the client actors are spawned by the server, executed on the client machine and the results are reported back to the server.



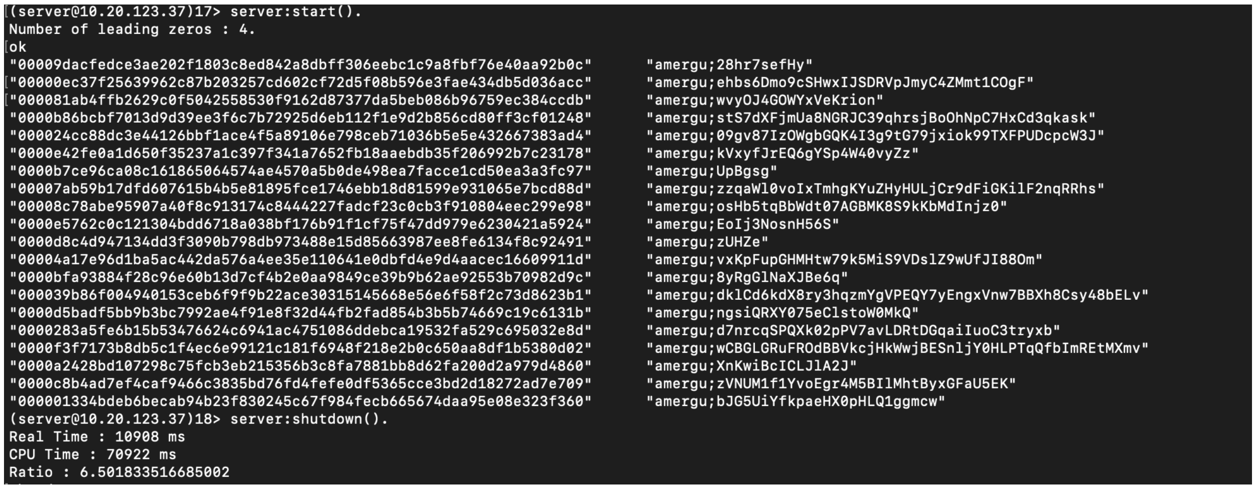
```
(client@rishi.local)15> server:client('10.20.123.37').
```

Size of the work unit for each worker:

The program bases its logic on the randomization. The initial work unit we implemented was the distribution based on the string lengths, with each worker working on a different string length to determine the random strings and then hash them. But the performance was better when the random string lengths are worked upon by each worker, than we assigning the string lengths for them. Performance vs the CPU utilization has reached the peak when the worker count is **10** for the program. A charset of lowercase and uppercase letters, and digits was taken.

The result of running program for 4 zeros:

Input leading zeros: 4

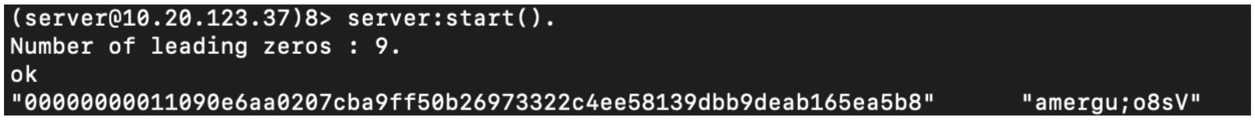


```
(server@10.20.123.37)17> server:start().
Number of leading zeros : 4.
ok
"00009dacfedce3ae202f1803c8ed842a8dbff306eebc1c9a8fbf76e40aa92b0c"      "amergu;28hr7sefHy"
"00000ec37f25639962c87b203257cd602cf72d5f08b596e3fae434db5d036acc"      "amergu;ehbs6Dmo9cSHwxIJSDRVpJmyC4ZMmt1COgF"
"000081ab4ffb2629c0f5042558530f9162d87377da5beb086b96759ec384ccdb"      "amergu;wvyOJ4GOWYxVeKrion"
"0000b86bcbf7013d9d39ee3f6c7b72925d6eb112f1e9d2b856cd80ff3cf01248"      "amergu;stS7dXFjmUa8NGRJC39qhrsjBoOhNpC7HxCd3qkask"
"000024cc88dc3e44126bbf1ace4f5a89106e798ceb71036b5e5e432667383ad4"      "amergu;09gv87IzOWgbGQK4I3g9tG79jxiok99TXFPUDcpcW3J"
"0000e42fe0a1d650f35237a1c397f341a7652fb18aaebdb35f206992b7c23178"      "amergu;kVxyfJrEQ6gYSp4W40vyZz"
"0000b7ce96ca08c161865064574ae4570a5b0de498ea7facce1cd50ea3a3fc97"      "amergu;UpBgsg"
"00007ab59b17dfd607615b4b5e81895fce1746ebb18d81599e931065e7bcd88d"      "amergu;zzqaWl0voIxTmhgKYuZHyHULjCr9dFiGKilF2nqRRhs"
"00008c78abe95907a40f8c913174c8444227fadcf23c0cb3f910804eec299e98"      "amergu;osHb5tqBbWdt07AGBMK8S9kKbMdInjz0"
"0000e5762c0c121304bdd6718a038bf176b91f1cf75f47dd979e6230421a5924"      "amergu;EoIj3NosnH56S"
"0000d8c4d947134dd3f3090b798db973488e15d85663987ee8fe6134f8c92491"      "amergu;zUHZe"
"00004a17e96d1ba5ac442da576a4ee35e110641e0dbfd4e9d4aacec16609911d"      "amergu;vxKpFupGHMHtw79k5MiS9VDslZ9wUfJI88Om"
"0000bfa93884f28c96e60b13d7cf4b2e0aa9849ce39b9b62ae92553b70982d9c"      "amergu;8yRgGlNaXJBe6q"
"000039b86f004940153ceb6f9f9b22ace30315145668e56e6f58f2c73d8623b1"      "amergu;dklCd6kdX8ry3hqzmYgVPEQY7yEngxVnw7BBXh8Csy48bELv"
"0000d5badf5bb9b3bc7992ae4f91e8f32d44fb2fad854b3b5b74669c19c6131b"      "amergu;ngsiQRXY075eClstoW0MkQ"
"0000283a5fe6b15b53476624c6941ac4751086ddebca19532fa529c695032e8d"      "amergu;d7nrcqSPQXk02pPV7avLDRtDGqaiIuoC3tryxb"
"0000f3f7173b8db5c1f4ec6e99121c181f6948f218e2b0c650aa8df1b5380d02"      "amergu;wCBGLGRuFROdBBVkcjHkWwjBESnljY0HLPTqQfbImREtMXmv"
"0000a2428bd107298c75fcb3eb215356b3c8fa7881bb8d62fa200d2a979d4860"      "amergu;XnKwiBcICLJlA2J"
"0000c8b4ad7ef4caf9466c3835bd76fd4fefe0df5365cce3bd2d18272ad7e709"      "amergu;zVNUM1f1YvoEgr4M5BIlMhtByxGFaU5EK"
"000001334bdeb6becab94b23f830245c67f984fecb665674daa95e08e323f360"      "amergu;bJG5UiYfkpaeHX0pHLQ1ggmcw"
(server@10.20.123.37)18> server:shutdown().
Real Time : 10908 ms
CPU Time : 70922 ms
Ratio : 6.501833516685002
```

The running time and real time for above program:

CPU time: 70922 ms
Real time: 10908 ms
Ratio of CPU time to REAL time: 6.501

The coin with the most 0s the program managed to find:



```
(server@10.20.123.37)8> server:start().
Number of leading zeros : 9.
ok
"00000000011090e6aa0207cba9ff50b26973322c4ee58139dbb9deab165ea5b8"      "amergu;o8sV"
```

Number of zeros: 9
String found: amergu;o8sV

The largest number of working machines you were able to run your code:

Number of machines: 4

Theoretically any number of client machines can be used. But, we have tried with 4 different machines.