**15-640 Lab 1**

Yin Xu (yinxu), Zheng Kou (zhengk)

1. **Design**:

Our communication model is based on the master/slave architecture. Master server acts as a user interface and a communication hub among every slave server. Slave servers are only responsible for execute commands that master server sends to them, such as starting a process, suspending a running process and resume a suspended process, etc. The migratable processes are process running in the JVM that extends the MigratableProcess Interface. By extending the MigratableProcess Interface, the processes can be controlled by the master server. Note that every slave server shares a ready-available distributed file system which allows them to read and write file in one file system instead of its own local disk. This will make our design simpler.

2. **Implementation:**

**Server Side:**

The master server is able to prompt the user to input command, parse it and execute it if valid.

Functions of master server includes:

1. Quit the program.

2. Print the hosts list.

3. Print all the processes on the each host and their status (running, suspending, terminated).

4. Start a new process on a specific host.

5. Migrate a running process to a specific host.

Unimplemented features of master server includes:

1. Balance the load of each slave server, so that they have similar amount of running processes.

2. Update the current alive hosts list when the master server lost its connections with some of the slave servers.

3. Check if the arguments that will be passed to the slave server is valid or not. In our implementation, the arguments verification only exists in slave side.

**Slave Side:**

Functions of slave servers include:

1. Start running a new process.

2. Suspend a running process and write the suspended process into a .obj file.

3. Resume a process by read the \*.obj file that previously dumped by another host.

4. Concurrently run multiple processes in one slave node.

Unimplemented features of slave server includes:

1. Terminated a running process once it lost its connection between the master server.

2. Being able to reuse the process ID number.

3. Notify the master once a running process is terminated. Now the master server must execute list process command in order to renew the status.  The server will not receive active notification from the slave when a process is done.

3. **Instructions (Build, Deploy and Run)**:

* **Build**
  + go to the folder containing this report;
* **Deploy**
  + start ProcessManager on the server:

java -cp ./bin ProcessManager

* + start ProcessManager on all the slaves:

java -cp ./bin ProcessManager -c <hostname or ip>

* **Run**
  + See Test Examples.

4. **System requirements:**

a.Java Virtual Machine

b. A shared distributed file system

5. **Test examples:**

**Example 1:** CaesarCipher

Description: Encrypt the input file using CaesarCihper strategy. Here, we simply move 3 positions of each character down the ascii table. To decrypt, move 3 positions back.

Usage: CaesarCipher <encode | decode> <inputfile> <outputfile>

Run:

Assuming you have started one master server and several slave servers.

a. to start a CaesarCipher process on slave #0:

start 0 CaesarCipher encode in.txt encoded.txt

   NOTE:  the whole encoding process takes approximately three minute to finish, so the user should **do all the migration in three minutes.**

b. to migrate the process, whose id is 1,  from slave #0 to slave #1:

migrate 1 0 1

c. migrate again, from slave #1 to slave #2:

migrate 1 1 2

d. you can also start more than one CaesarCipher processes and migrate them.

e. after the first process finished, you might want to start a decode process. Since all servers are sharing the same file system, you can do this on any of them. Here we use slave #1:

start 1 CaesarCipher decode encoded.txt decoded.txt

d. to migrate the decryption process from slave #1 to slave #0

migrate 2 1 0

e. finally, when you see “Finished processing” notification in slave #1,  you can check in.txt and decode.txt, which should be identical.

**Example 2:** FileCompression

Description: Compress the input file using GZIP compression stream.

Usage: FileCompression <inputfile>

The output compressed file name is <inputfile>.gz by default.

Run:

a. to start a FileCompression process on slave #0:

start 0 FileCompression in.txt

   NOTE:  the whole encoding process takes approximately two minute to finish, so the user should **do all the migration in two minutes.**

b. the migration instructions are exactly the same as example 1.

c. when done, the slave terminal will notify “Finished processing”.  A compressed file named in.txt.gz will be located in the same directory.

d. To verify the compression result, you can use the below command to decompress the file and compare it with the original input file:

gizp -d in.txt.gz