15-440 Lab 1

Pan Sun (pans), Dengbo Wang (dengbow)

Design:
Our model has a process manager server and several worker servers. It is closed to a real distributed system like map-reduce system.
Manager server acts as an user interface and a communication hub. It must be started before any worker server. It has a foreground thread as a console, a background thread that handles new connections, and several background communication threads that receive messages from workers. The console allows you to manage process such as creating, suspending, migrating, resuming and displaying processes.
Worker servers are only responsible for execute commands that manager server sends to them, such as starting a process, suspending a running process and resume a suspended process, etc. Each worker can assigned several tasks and will send back process status message. It has several threads that run the processes and a thread that listens to the socket and a thread that checks the status of every process.
The migratable processes are processes running in the JVM that implements the MigratableProcess Interface. By implementing the MigratableProcess Interface, the processes can be controlled by the manager server. Note that every worker server shares a distributed file system which allows them to read and write file in one file system. Each process has a unique ID associated with its status.

Features:
1. There two ways to migrate a process: First, you can combine suspend and resume commands; Second, use migrate command.
2. Using concurrentHashMap allows running of multiple processes at the same time, avoid race condition.
3. Using design pattern such as decorator, singleton, strategy, template to achieve low coupling, high cohesion and so on.

Implementation:
Process Manager:
1. Quit the system and shut down all servers
   quit
2. List all processes with their status and workers
   worker
   process
3. Show the help message
   help
4. Create a new process and run it on a specific worker
   start <workerID> <processName> <arg1> <arg2>…..
   e.g. start 0 Code encode file/p1.txt file/encoded.txt….
5. Suspend a specific process and serialize it
   suspend <processID>
6. Resume a suspended process with any worker
   resume <processID> <workerID>
7. Migrate a suspended or running process from one worker to another
   migrate <processID> <workerSrcID> <workerDesID>
8. ConnectionListener: handle new connection from a worker server.

Worker:
1. Start 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 worker.
5. Check for completeness of any process.
6. send back messages to manager server.

Unimplemented features and bugs:
1. We don't have any intelligence to balance the processes between workers.
2. Can't react to the loss of a worker.
3. Manager can't verify the status of any process that is assigned to any worker.
5. As the number of processes increases, migration needs more time. Sometimes it may leads to error because start command is executed before the generation of .obj file.
6. the process ID keeps increasing and we can't reuse them.
7. We can't reuse threads.
8. We can't update the list of workers.

How To Run, Build, Deploy:
1. Change the manager port in ProcessManager class manually if the original one doesn't work.
2. Change the directory in ProcessSerialization class manually to a proper location if possible.
3. Go to the folder containing the src folder and the report.
4. run the script on all severs to compile and build the program:
   ```
   bash make.sh
   ```
5. run the script on the processManager:
   ```
   bash run.sh
   ```
6. run the script on all the workers:
   ```
   bash run.sh <manager IP address or name>
   ```
7. After you quit the system using quit command, run the script on servers to clean the file system:
   ```
   bash clean.sh
   ```

System requirements and dependencies:
1. A shared distributed file system
2. All connections will not be lost, all workers never leave.
3. you have correctly configured the port and directory.
4. There are not too many processes to run.

Tests:
Test 1: Code
Description: Encrypt and decrypt the input file to destination file.
Usage: Code <encode | decode> <inputfile> <outputfile>
Run:
1. Ensure that you have correctly set the parameters.
2. you have finished How To Run part.
3. create a process on worker 0. It takes 2 minute.
   ```
   start 0 Code encode file/p1.txt file/encoded.txt
   ```
4. Migrate the process 0.
   ```
   migrate 0 0 1
   ```
5. suspend the process 0.
   ```
   suspend 0
   ```
6. resume the process 0 on worker 1.
   ```
   resume 0 1
   ```
7. Create another Code process and migrate it.

```
start 1 Code decode file/encoded.txt file/decoded.txt
migrate 1 1 0
```

8. you will see “Finished Processing0” on worker 1.
9. Check the correctness.
10. quit the program on the manager server.

```
quit
```

11. bash clean.sh

Test 2: FileZipper
Description: Compress the input file using GZIP compression stream.
Usage: FileZipper <inputfile>
Output compressed file name is <inputfile>.gz by default.
Run:

1. start 0 FileZipper file/p1.txt. It takes 3 minute.
2. migrate the process.
3. the result is like Test 1.
4. Check it by comparing the decompressed file with the original file.

Test 3: PrintNums
Description: print number from 1 to 30.
Usage: PrintNums

Test 4: GrepProcess
Description: query string for every line.
Usage: GrepProcess <queryString> <inputFile> <outputFile>

For all examples, you can use whatever command explained above.