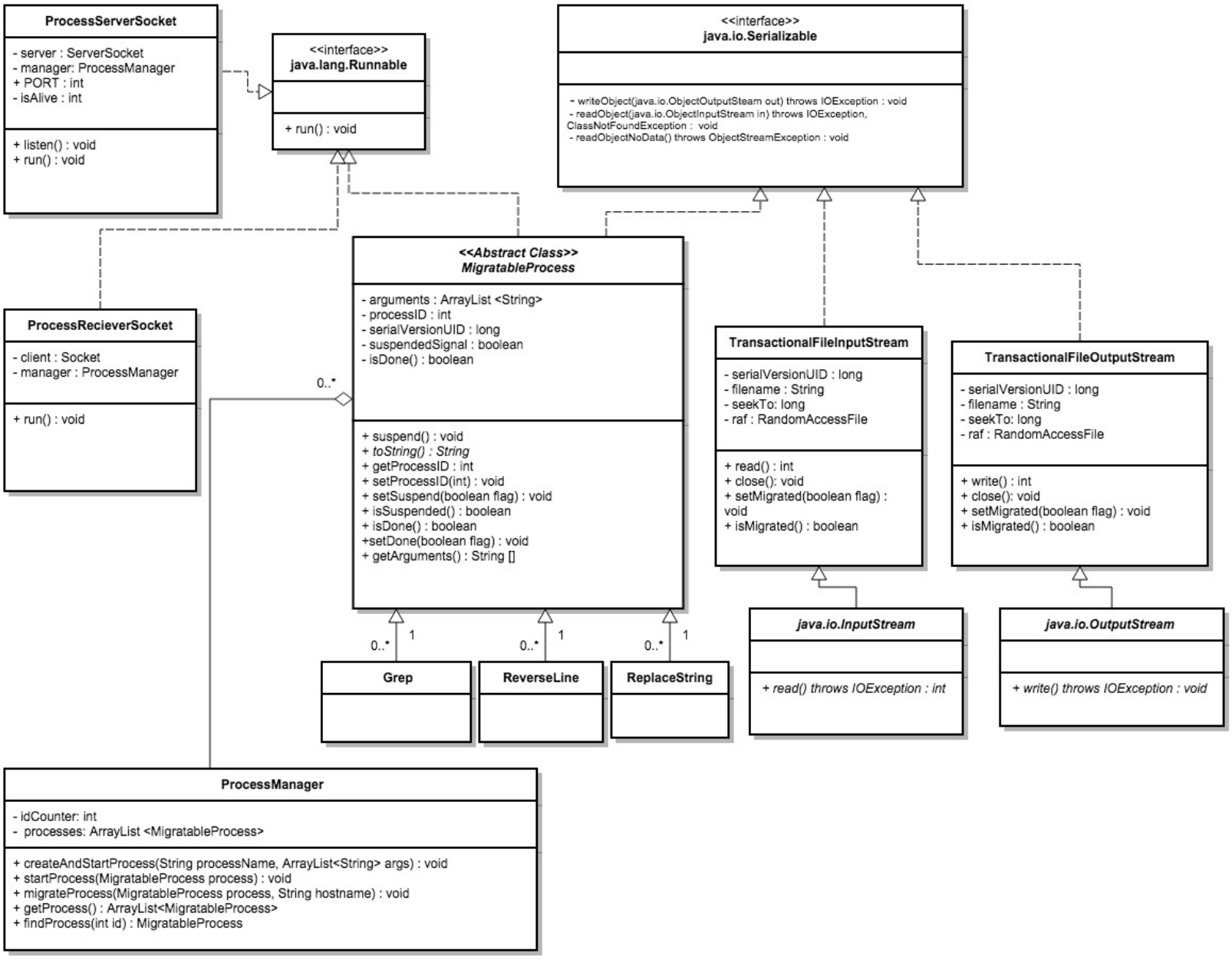
15-440 Project 1

MIGRATABLE PROCESSES

Shri Karthikeyan (skarthik),
Nandini Ramakrishnan (nramakri)

Table of Contents:

1. Class Diagram



2. High-level description of design

The problem that we are trying to solve with this lab is ***Process Migration***. Process migration refers to the situation when a process is transferred from one computer to another, but the process must be unaware of it (i.e. the process must continue executing as usual). The “Migratable Processes” project demonstrates how we can achieve this.

Our project architecture is such that every node has its own process manager that keeps track of all the processes. Each node also has a ServerSocket that listens for incoming migratable processes. The example problem flow in Section 6 gives a detailed explanation of how the components work together to migrate processes.

Our program gets executed when the user enters a command on the command-line. The user can give commands such as to start and migrate processes from his node. This command-line is read by the class ProcessRunner. It is then parsed, and depending on the command, it calls the respective method from the ProcessManager class. For a detailed account of how Process Runner and ProcessManager work together, refer to Section 3 (Description of the Components).

All migratable processes conform to the model specified in an abstract base class named MigratableProcess. The reason for choosing an abstract class instead of an interface was for code re-use. Another design choice we incorporated in this project, was to leave it to the process to inform the ProcessManager when it terminated.

3. Description of components

ProcessRunner

This class launches the command-line, parses the given command, interprets the desired action and forwards it to the ProcessManager. The ProcessRunner has additional methods, “help”, to display information to the user about the program, and “print”, which prints the currently running processes on a particular node.

ProcessManager

The ProcessManager is the heart of the project. It executes the commands given to it by the ProcessRunner. It has four methods, which are as follows:
createAndStartProcess - This creates the process given by ProcessRunner, and starts its execution
startProcess - This resumes a process which was suspended due to migration
migrateProcess - This method carries out migration from the sender. It serializes the object and writes it onto the output stream of a ProcessServerSocket.

ProcessServerSocket

This class represents the sender node of the migratable process. This object is essentially a socket which is continuously listening for incoming connections. Once a connection is accepted, it spawns a new thread, which will run ProcessReceiverSocket.

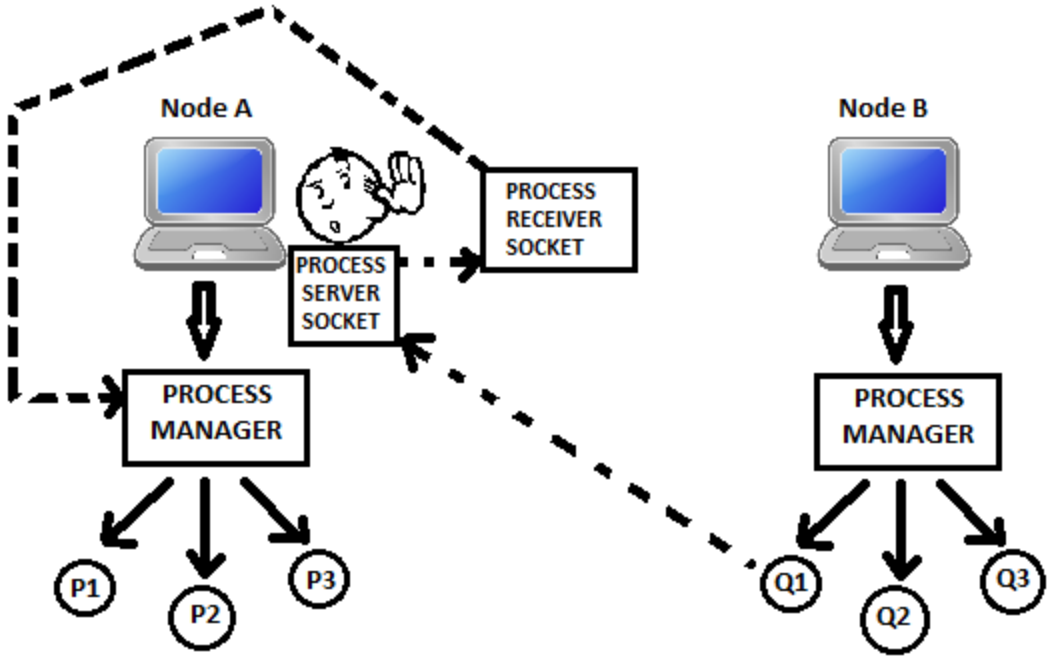
ProcessReceiverSocket

This class represents the receiver node of the migratable process. Like ProcessServerSocket, ProcessReceiverSocket is also essentially a socket, which is receiving the serialized data sent from ProcessManager. It reads the socket’s input stream, de-serializes the data and runs the startProcess method of ProcessManager to continue execution of the migrated process.

Transactional I/O Library

Two classes have been provided i.e. TransactionalFileInputStream and TransactionalFileOutputStream. These two classes implement Serializable and override the methods “read” and “write”. For the purposes of serialization, any read and write methods involved will call the Transactional I/O Library’s read and write methods. These methods have the added capability of checking whether the particular file is null or whether the current process may have migrated.

4. Example program flow



In the above diagram, Process Q1 from Node B, needs to migrate to Node A. It tries to connect to the ProcessServerSocket on Node A. ProcessServerSocket is always listening for incoming connections. This accepts the connection and forwards the incoming data to ProcessReceiverSocket on Node B. ProcessReceiverSocket will de-serialize the data and redirect the data to the Process Manager of Node B, which restarts the suspended process.

5. Test programs

1) ReverseLine: This program reads every line of a file, reverses it and prints it onto an output file

2) ReplaceString: This program read every line of a file, finds a particular given string

ReverseLine Execution: Testing output

Sender node:

```
[nramakri@unix5 src]$ java edu/cmu/cs/cs440/hw1/ProcessRunner
Hi! Please run all of you migratable processes here!
Type 'help' to find out the commands!
> start edu.cmu.cs.cs440.hw1.ReverseLine testFiles/inputText.txt testFiles/outputText.txt
> print
[ReverseLine:ProcessID:0:Arguments:[testFiles/inputText.txt, testFiles/outputText.txt]]
> migrate 0 unix2.andrew.cmu.edu
> exit
Exiting...
Goodbye!
[nramakri@unix5 src]$ java edu/cmu/cs/cs440/hw1/ProcessRunner
Hi! Please run all of you migratable processes here!
Type 'help' to find out the commands!
> start edu.cmu.cs.cs440.hw1.ReverseLine testFiles/inputText.txt testFiles/outputText.txt
> print
[ReverseLine:ProcessID:0:Arguments:[testFiles/inputText.txt, testFiles/outputText.txt]]
> migrate 0 unix2.andrew.cmu.edu
> exit
Exiting...
Goodbye!
```

Receiver Node:

```
[nramakri@unix2 src]$ java edu/cmu/cs/cs440/hw1/ProcessRunner
Hi! Please run all of you migratable processes here!
Type 'help' to find out the commands!
> There is a connection trying to be made
The process is Migrating..
Migrated!
pr
> print
[ReverseLine:ProcessID:0:Arguments:[testFiles/inputText.txt, testFiles/outputText.txt]]
> print
[]
> ^[
```

Input text file:

```
[nramakri@unix5 src]$ cat testFiles/inputText.txt
Shri and Nandini
Hot Chocolate
Water bottle
Cell Phone
Harry Potter
Andrew Carnegie
[nramakri@unix5 src]$
```

Final output text file (after migration):

```
[nramakri@unix2 src]$ cat testFiles/outputText.txt
inidnaN dna irhS
elttob retaW
enohP lleC
rettoP yrraH
eigenraC werdnA
genraC werdnA
[nramakri@unix2 src]$
```

6. How to run the project

Our project contains one main folder src, which contains the folder “edu” and test files. We have placed two text files in the src folder: inputText1.txt and inputText2.txt. You can add any text file to test in this directory.

Once you are in src, run the following set of commands:

```
> javac edu/cmu/cs/cs440/hw1/*.java
> java edu/cmu/cs/cs440/hw1/ProcessRunner
```

On successfully launching the project, you should receive a welcome message on the command-line: These are the commands that you can use:

1) *Start*
To start any other generic migratable process, create an instance implements the methods of the abstract base class MigratableProcess and put it into the src folder.

```
> start <class-name> <input-file> <output-file> <arg_0> … <arg_n>
```

For example,
to start “ReverseLine”, type

```
> start edu.cmu.cs.cs440.hw1.ReverseLine inputText1.txt outputText1.txt
```

where `inputText1.txt` is the file you read and `outputText1.txt` is the file the output is written onto.

and to start “ReplaceString”, type

```
> start edu.cmu.cs.cs440.hw1.ReplaceString inputText2.txt
  outputText2.txt Hello Goodbye
```

where `“Hello”` is the string you find, `“Goodbye”` is the string you replace it with, `inputText2.txt` is the file you read and `outputText2.txt` is the file the output is written onto.

2) *Migrate*

To migrate a process, you require its process ID and the hostname of the destination node.

```
> migrate <process-ID> <host-name>
```

For example,

```
> migrate 0 unix1.andrew.cmu.edu
```

3. *Help*

To get additional information about the command-line functions, type help at the command-line.

```
> help
```

4. *Print*

To see the currently running processes on a node, type print.

```
> print
```

Dependencies

Java Version 1.7 is preferred