# Lab 1: Process Migration

Due: 1/29/09

January 15, 2009

## 1 Introduction

Process migration refers to a system where running processes can be moved from one node to another over the course of their life. A process should be agnostic to what node it is running on and should be unaware of whether it has been moved from one node to another. Thus, the process is provided with the illusion that it is running on a single node during its entire execution. To achieve this, it is necessary to be able to pause and package up a process, ship it to another node, and unpackage and resume it such that it is running again. The process should not lose it's location in the program, variables, open files, network connections or any other state.

In this lab, you will work with a partner to write a basic process migration system in Java.

## 2 Important Dates

-Lab Due: 11:59:59pm EST, January 29th, 2009

## 3 Lab Requirements

In this lab you will need to create a utility for migrating java processes. This utility, implemented as the `ProcessManager`, will need to launch new processes and migrate processes from one node to another in order to balance load. Additionally, you will be required to implement two migratable processes.

## 3.1 Migratable Processes

For this lab we will focus our attention on processes that are specially built to be migratable. To that end, we will look only at processes that implement the `MigratableProcess` interface. The `MigratableProcess` interface extends `java.lang.Runnable` (allowing it to be run via a `java.lang.Thread` object) and the `java.io.Serializable` interface to permit it to be serialized and writen to or read from a stream (see Section 3.6). The interface requires a `void suspend(void)` method which will be called before the object is serialized to allow an opportunity for the process to enter a known safe state. A `String toString(void)` method is also required. This method should print the class name of the process as well as the original set of arguments with which it was called. Furthermore, all `MigratableProcess`es must include a constructor which takes, as it's sole argument, an array of strings. Finally, the process will limit it's I/O to files accessed via the `TransactionalFileInputStream` and `TransactionalFileOutputStream` classes, discussed in Section 3.2.

## 3.2 Transactional I/O

To facilitate migrating processes with open files, you will need to implement `TransactionalFileInputStream` and `TransactionalFileOutputStream`. These classes will extend `java.io.InputStream` and `java.io.OutputStream`, respectively, as well as implement the `java.io.Serializable` interface. When a read or write is requested via one of these classes, they should open the file, seek to the requisite location, perform the operation, and close the file again. In this way, they will maintain all the information they require in order to continue performing operations on the file even if the process is transferred to another node. Note that you may assume that all of the nodes share a common filesystem, such as afs, where all of the files to be accessed will be located.

## 3.3 Launching Processes

The `ProcessManager` monitors standard in for requests to launch processes. These requests take the form of "<processName> [arg1] [arg2] ... [argN]". When a request is made, a new object of class <processName> should be instantiated and all of the arguments should be passed in an array to the constructor. If an invalid class is named (does not exist or does not implement `MigratableProcess`), then an appropriate error message should be printed and operations should continue as normal.

Do not hard-code support for mapping particular known strings to particular classes. Your ProcessManager should be able to handle any MigratableProcess, not just the ones you have implemented. This means that you will not know what class you are instantiating until runtime. Thus, you will likely need to use the java.lang.Class<T> class and java.lang.reflect.Constructor<T> class. In particular, the following functions are likely to be of use:

- Class.forName(...)
- Class.getConstructor(...)
- Constructor.newInstance()

Note well: The java class names for arrays are not what you might expect. For example, the class name for an array of strings (String[]) is "[Ljava.lang.String;".

### 3.4 Accepting Commands

Your ProcessManager should create a provide a prompt on which commands can be entered. The following commands must be supported:

- <processName> [arg1] [arg2] ... [argN] (see Section 3.3)
- ps (prints a list of local running processes and their arguments)
- quit (exits the ProcessManager)

In addition to supporting these commands, your ProcessManager should print a message when a process is terminated.

An interaction with the ProcessManager might look something like this:

```
==> ps
no running processes
==> TestProcess foo.dat 1000
==> TestProcess bar.dat 50
==> ps
TestProcess foo.dat 1000
TestProcess bar.dat 50
==> ps
TestProcess foo.dat 1000
Process "TestProcess bar.dat 50" was terminated
==> quit
```

Finally, you are welcome to add other commands if you wish.

### 3.5 `ProcessManager` Arguments

If the `ProcessManager` is supplied with the arguments "-c <hostname>" then the `ProcessManager` should run as a *slave* and connect to the *master* running on <hostname>. If no such argument is supplied, then the `ProcessManager` should run as the *master*. You are welcome to support other arguments if you wish. In particular, it may be of value to have an argument for selecting a port to use for communications between the master and the slaves.

### 3.6 Migrating Processes

One of the running `ProcessManager` instances will be designated the *master* instance. This instance is required to query the other instances for their load and migrate processes to balance the load as well as possible. These queries should be made at a rate of once every 5 seconds. For the purposes of this lab, the load on a node is defined to be the number of migratable processes currently running on it.

In order to migrate a process, it will be necessary to move the `MigratableProcess` object from one `ProcessManager` to another. To accomplish this, you will want to make use of the `java.io.ObjectOutputStream` and `java.io.ObjectInputStream` classes.

### 3.7 Implemented Processes

In addition to writing the `ProcessManager`, you will write two different `MigratableProcess`es that run under your system. These need not be long complex programs, but they should be interesting and should involve I/O. A few suggestions include: zip, image processing such as edge detection, and a basic webcrawler. If you would like to do something else, please check your idea with us.

## 4 Technical Requirements

- Your project must function on the linux.andrew.cmu.edu pool of computers
- Your code must compile properly in Java 1.5.0
- Your project must include a Makefile that builds your project

# 5 Grading

25% code quality, style

55% functionality and robustness of `ProcessManager`

20% functionality of implemented processes

# 6 Handin

Projects are to be handed in electronically in the following directory:
`/afs/andrew.cmu.edu/course/15/440-sp09/handin/lab1/`

You should create a directory there that consists of the hyphenation of your andrew IDs (e.g. gkesden-mpa or jdkaufma-jzaman) and drop your code inside. If, for whatever reason, you wish to re-handin your code, simply create a new directory with the same name as before followed by .<version number> (e.g. gkesden-mpa.1). You may submit as many versions as you would like. We will grade only the one with the highest version number.

# 7 Nota Bene

- Find a partner right away. If you need help, let us know.
- Set up and use version control. Now.
- Settle any major style wars with your partner before starting to write code. Inconsistent code is painful to read will be reflected in your grade.
- Document your code (and its bugs) as you write it. Do not put documentation off until the last minute.
- Think/write test code for your process migration utility concurrently with developing your process migration utility. How will you know it works?
- Set aside time to meet regularly with your partner.
- Remember to ask for help sooner rather than later.