Bachelor of Science (Hons)
Project Proposal
Department of Computer Science
Rhodes University
Grahamstown

Author: D.G. Smith
Supervisor: Prof. G. C. Wells

26 February 2016

1 Principle Investigator

Dylan Gregory Smith (13s0714)
14 Dulverton Road
Somerset Heights
Grahamstown, 6139

Supervised by Prof G. C. Wells (g.wells@ru.ac.za)

2 Project Title

The following project title is proposed to the Department of Computer Science, Rhodes University:

Interprocess Communication with Java in a Microsoft Windows Environment - Heading Towards a Generic IPC Design.

3 Problem Statement

Java[1] is a widely used general-purpose programming language that supports an object-oriented programming model as well as features such as multithreading, portability and simplicity (Oracle, 2010). Java currently lacks support for interprocess communication (IPC) within differing address spaces (distinct from threads) but instead relies on a distributed network programming mechanism which introduces significant overhead (Wells, 2010). The Linux and Solaris IPC implementations exist as a starting point for research into the development of a Windows implementation and eventual generic Java class library that will appear to be platform independent to the Java programmer (i.e. avoid portability issues of native code). The native code that is used will still be machine dependent. As a result, this requires significant research into the internal Windows IPC structures as well as other OS's IPC functionality to determine any commonalities that may assist in the development.

4 Research Objectives

The primary research objective is to understand how the Microsoft Windows operating system implements its IPC. This will involve the mastering of the Java Native Interface (JNI) framework in order to access lower-level OS facilities. Emphasis is placed on the implementation of this to provide Java with concurrency mechanisms.

Secondary objectives include a thorough investigation of other OSs' (such as Linux, Mac OS and so forth) IPC facilities with the eventual aim of designing a generic Java class library that can be used on a range of OSs. Time permitting, a prototype will be designed, with test programs being used to analyse the results.

5 History and Background

Processes and threads within all modern computer operating systems have become a significantly important abstraction and without such, modern computing would not exist as we know it today. These aspects are fundamental to operating system design and implementation and have an important effect on efficiency and operability of the OS (Tanenbaum and Bos, 2015). When programs are executed, a process is created which contains the code and the data which the program is using. The process is then terminated when the program is terminated (Carver and Tai, 2006). Multicore processing is steadily becoming ubiquitous in modern computing; present in most modern day desktop machines. Most machines no longer rely on a uniprocessor model (Hayes, 2007). As such, the need for concurrency and communication amongst processes running in separate ad-

[1] http://www.oracle.com/technetwork/java/javase/overview/index.html

dress spaces is vital in achieving parallelism.

Interprocess communication refers to the mechanism of communication between programs executing within a shared-memory environment but running within different address spaces (Wells, 2009). Initial research conducted by Wells (2010) investigated concurrency in Java, and led to the observation of a lack of communication mechanisms offered by the Java programming language. In contrast, Java provides a good method in which to implement multithreading programming using `java.util.concurrent`[2] but not communication, and hence synchronisation with programs running in separate Java Virtual Machines (JVMs).

The support that is provided is by means of a "loopback" network connection. This approach can be significantly inefficient, as it forces communication to traverse the layers of the protocol stack (Wells, 2009).

The Java Native Interface (JNI) is a framework that is part of the Java platform that can allow the access of native code written in languages such as C and C++ (Liang, 1999). This means that low-level OS features (memory, I/O and so forth) can be accessed using the JNI framework by making use of native code (Dawson et al., 2009). The goal here is to make use of JNI's functionality to gain access to low-level operating system features to design a more efficient IPC solution for two or more Java programs running in separate JVMs, rather than using the inefficient "loopback" solution. As mentioned previously, a Linux version of this exists as a starting point for research into the design of a Windows implementation and eventual generic class library design. The following code illustrates how a Java method is declared as `native` which means its actual implementation will be in C/C++:

```
public native String stringMethod(String text);
```

When this code is compiled using `javah`, a C header file is generated that defines a function header for the Java function as illustrated below:

```
JNIEXPORT jstring JNICALL Java_Sample1_stringMethod
(JNIEnv *, jobject, jstring);
```

This function can then be implemented in a C source code file and then compiled using any standard C compiler. Care must be taken to ensure that method signatures match. A shared library file (.dll for Windows) will be created. The Java program can then be executed (using `java`), calling the native C code that was implemented[3]. Since C code can be used by means of JNI, it may be possible to use this as a mechanism to access lower-level OS features, hence providing an opportunity to enhance Java's IPC facilities.

[2] http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html

[3] Example adapted from: http://www.ibm.com/developerworks/java/tutorials/j-jni/j-jni.html

In order to achieve this in terms of Microsoft Windows, a thorough understanding of the internal IPC functions are required such as `CreateProc`[4]. Windows' executive process (EPROCESS) blocks will also have to be investigated and how it points to other data structures relating to IPC (Russinovich and Solomon, 2009).

6 Approach

The first phase involves the investigation and understanding of Windows internals and how the operating system implements IPC. This will involve the analysis of existing literature and code libraries that discuss this. IPC in other operating systems will also be investigated. Analysis of the provided implementation in Linux will also aid in gaining familiarity. Further investigation of how Java's JNI framework functions and how the C programming language can be used will also take place. Fairly small programs will be written to test aspects such as level of access, effectiveness and efficiency

The second phase will involve the actual Windows implementation, making use of the analysis of JNI and the Linux version.

The third phase will involve performance benchmarking. Simple test programs will be used to compare the implementation against Java's loopback solution. This will provide insight into the efficiency of the implementation

With time permitting, a design of a generic class library will be completed with the eventual development of a prototype that can be tested.

7 Proposal Conclusion

This project in essence falls within parallel programming with the Java programming language within the context of multicore processing and addressing the lack of coordination in Java's communication capabilities. It also requires a substantial amount of research of different operating system's internal design principles in terms of IPC functionality in order to aid in the development of a generic class library. Ultimately the use of the Java Native Interface will be used in this development process. At the conclusion of the research, simple test programs will be written and executed in order to determine the effectiveness of the solution and to establish a sufficient benchmark.

[4]https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425%28v=vs.85%29.aspx

9 Timeline

Project Activity	Proposed Deadlines
Conduct initial investigation into literature	22 February 2016
Compose and submit proposal	26 February 2016
Seminar 1	1 March 2016
Acquire sufficient proficiency in Java	12 March 2016
Master the JNI framework	1 April 2016
Attain an understanding of Windows internals	15 April 2016
Experiment with JNI and C with internal structures	29 April 2016
Attain additional necessary literature	9 May 2016
Literature Review - first draft	23 May 2016
Literature Review - final submission	27 May 2016
Code design for IPC access for Java	8 July 2016
Seminar 2	26 July 2016
Code implementation	12 August 2016
Testing and benchmarking	22 August 2016
Possible investigation of other OS IPC mechanisms	29 August 2016
Short paper - first draft	7 September 2016
Short paper - final submission	12 September 2016
Design of generic Java class library	23 September 2016
Thesis - first draft	17 October 2016
Seminar 3	24 October 2016
Thesis - final submission	28 October 2016

Note: the proposed dates are provisional and tasks may run concurrently.

8 Requirements

Software requirements are listed below:

- Microsoft Windows (XP and above).
- A Linux distribution (preferably Ubuntu 15.10).
- Java SE Development Kit (JDK).
- C Compiler, i.e. cc, gcc, cl (Windows).
- Microsoft Visual Studio 2015 Enterprise
- Various code editors (i.e. Visual Studio Code, Sublime, UltraEdit among others).

References

Carver, R. H. and Tai, K. (2006). *Modern Multithreading: Implemeting, Testing, and Debugging Multithreaded Java and C++/ Pthreads/Win32 Programs*. John Wiley & Sons, New Jersey.

Dawson, M., Johnson, G., and Low, A. (2009). Best practices for using the Java Native Interface. http://www.ibm.com/developerworks/library/j-jni/. [Date Accessed: 21 February 2016].

Hayes, B. (2007). Computing science: Computing in a parallel universe. *American Scientist*, 95(6):476–480.

Liang, S. (1999). *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley Professional, Palo Alto.

Oracle (2010). Introduction to the Java Programming Environment. https://docs.oracle.com/cd/E19455-01/806-3461/6jck06gqb/index.html. [Date Accessed: 23 February].

Russinovich, M. E. and Solomon, D. A. (2009). *Windows Internals Covering Windows Server 2008 and Windows Vista*. Microsoft Press, 5 edition.

Tanenbaum, A. S. and Bos, H. (2015). *Modern Operating Systems*. Pearson, Edinburgh, 4 edition.

Wells, G. (2009). Interprocess communication in Java. In Arabnia, H., editor, *Proc. 2009 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'09)*, pages 407–413, Las Vegas. CSREA Press.

Wells, G. (2010). Extending Java's communication mechanisms for multicore processors. In *8th International Conference on the Principles and Practice of Programming in Java (PPPJ 2010)*, Vienna, Austria. (Poster).