Advanced Science and Technology Letters
Vol.85 (Information Technology and Computer Science 2015), pp.68-71
http://dx.doi.org/10.14257/astl.2015.85.16

A Study on the Native Interface for Smart Virtual Machine

Yunsik Son[1], Jaehyun Kim[2], Yangsun Lee[2*]

[1] Dept. of Computer Engineering, Dongguk University
26 3-Ga Phil-Dong, Jung-Gu, Seoul 100-715, Korea
sonbug @dongguk.edu
[2] Dept. of Computer Engineering, Seokyeong University
16-1 Jungneung-Dong, Sungbuk-Ku, Seoul 136-704, Korea
{statsr, yslee}@skuniv.ac.kr
*Corresponding Author

Abstract. Although the executing program on virtual machine environment has an attribute of platform-independence in general, it shows low speed performance and it is unable to implement platform independency directly. Also it has a disadvantage that the library and program written in native language cannot be reused.

This paper presents the research about providing a native interface of Smart Virtual Machine (SVM). The paper also presents the operating mechanism of Java Native Interface (JNI), the interface generator, the interface definition language and so on to account for the native interface regulation.

Keywords: Native interface, Interface definition language, Language interoperability, Smart Virtual Machine

1 Introduction

As the virtual machine, the software product, is a conceptual computer which has logical system configuration, the executing program on virtual machine environment has an attribute of platform-independence. Due to the command execution system based on software, however, it shows low performance compared to the hardware based native execution system and it is unable to reuse the previously developed libraries and codes for native. Since the native interface is a technique to use the native library on virtual machine environment, implementing the feature which connects the execution environment of virtual machine and the real native operating system and defining the agreements to call native libraries on program are necessary [1], [2], [3]. This paper presents the research on providing a native interface on SVM.

2 JNI(Java Native Interface)

JNI is the native interface provided by JAVA. Developers can implement platform independent features or create java programs using the already existing libraries or programs written in other languages [4]. Also the speed improvement is expected by writing codes which can affect the performance of program significantly in native language. The way of writing and implementing a native method can be divided into 5 steps as shown in figure 1: the class file generation through the compile process, the header file generation using Java utility, the method implementation in native language and the dynamic library file generation [5], [6].

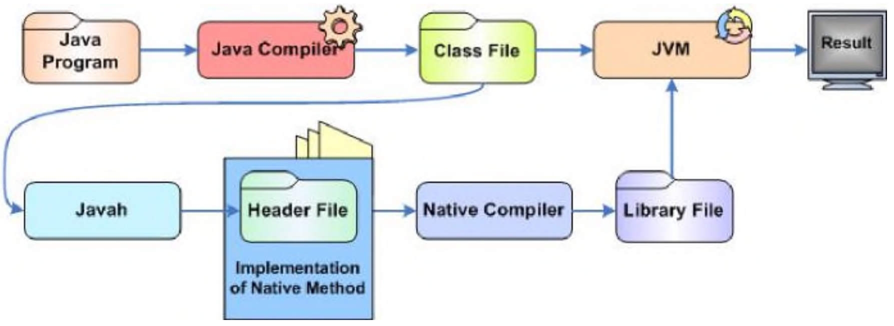

Fig. 1. Native code process model using Java native interface

3 Function Call Process

On the physical system such as x86 and on the conceptual system consisted of software, various cases which happen during function calls have to be considered. When the function call occurs, a caller forwards the parameter for the called function and saves the information including program counter, register, stack and heap to restore the current state at a point of called function expiration. After saving all the information for state restore, the caller changes the control to the called function through the program counter. A callee generates a data structure for the called function on memory and allocates the received parameter. After expiring the process for the function, the callee transmits the return value to the caller and the caller proceeds the restoration to the previously saved state and saves the return value received from the caller.

The way of use the register architecture, stack, heap and etc. for each system is different. The register doesn’t exist especially on the virtual machine as it designed based on stack in general. Therefore, the register or the memory area needed for performing the explained task has no choice but to shows the difference depending on system and language. Even though there is a few difference takes place during the process, the task mentioned above have to be all performed internally.

4 Interface Definition Language

Interface definition language is a name of language which enables programs or instances written in one language to communicate with other programs written in unknown language. The main purpose of use is to describe the interface provided by the server-side instances for the communication among instances apart from each other, and this provides an independent interface for both operating system and programming languages and a portability among different networks, different kinds of computers and different operating systems. The interface definition language can define more than one interface. Since the program written in the interface definition language only defines the set of functions or methods, it becomes independent from the programming languages which implement the structural elements and it enables to separate the part of implementation and interface. Interface definition language and compiler provides the clarity of implementing the interface, the software productivity improvement, the consistency assurance and the ease of maintenance to the programmers who are proficient to the existing programming language. Interface definition language is generally used for Remote Procedure Call (RPC) software as it provides the way to connect between two different systems. There are Microsoft Interface Definition Language (MIDL), Open Service Interface Definitions (OSID) and so on for the representative interface definition language.

5 Conclusions and Proposed Model

This paper analyzed such as the existing structure of JNI, the function call process and the characteristics of IDL to implement the native interface on SVM. The agreements of function calls for connection between virtual machines and native languages and the definition of table for the native function connection is necessary to ordain the native interface agreements on SVM.

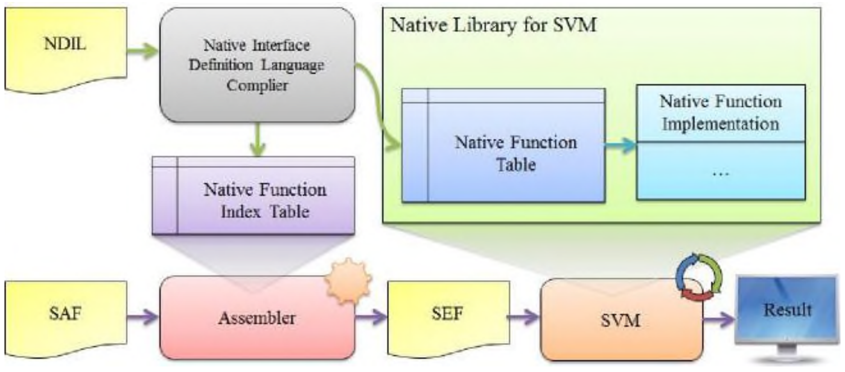
NDIL
Native Interface Definition Language Complier
Native Library for SVM
Native Function Table
Native Function Implementation
...
Native Function Index Table
SAF
Assembler
SEF
SVM
Result

Fig. 2. Native code process model with Native interface definition language for Smart Virtual Machine

Also, the model for tables and native functions is generated automatically to reduce the burden of table generation by using native definition language. Assembler, the execution engine performs tasks related to native functions by using the generated tables. The proposing native interface model of SVM based on this is on figure 2.

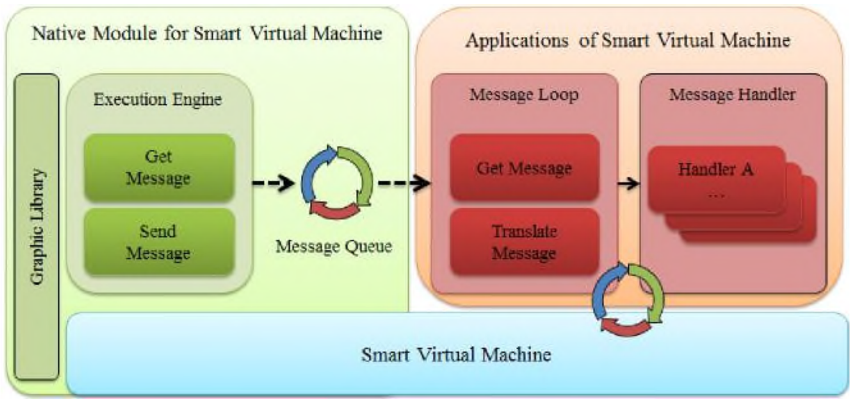

Fig. 3. Native code execution model in Smart Virtual Machine

Figure 3 shows the native code execution model using proposed native interface model. After this work, we are planning to expand the SVM based on the proposing model and then test the native interface to check the result.

Acknowledgments. This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and Future Planning (No.2013R1A2A2A01067205).

References

1. Y.S. Lee, Y.S. Son, "A Study on the Smart Virtual Machine for Executing Virtual Machine Codes on Smart Platforms", International Journal of Smart Home, SERSC, Vol. 6, No. 4, 2012, Australia, pp. 93-105.
2. Y.S. Lee, Y.S. Son, "A Study on the Smart Virtual Machine for Smart Devices", Information -an International Interdisciplinary Journal, Vol. 16, No. 2, International Information Institute, 2013, Japan, pp.1465-1472.
3. Y.S. Lee, S.M. Oh, Y.S. Son, "Design and Implementation of HTML5 based SVM for Integrating Runtime of Smart Devices and Web Environments", International Journal of Smart Home, SERSC, Vol.8, No.3, 2014, Australia, pp.223-234.
4. M. Campione, Kathy Walrath and Alison Huml, The Java Tutorial Continued: The Rest of the JDK, Addison Wesley, USA, 1998.
5. S. Liang, The Java Native Interface: Programmer's Guide and Specification, Addison Wesley, USA, 1999.
6. R. Gordon, Essential JNI: Java Native Interface, Prentice Hall, USA, 1998.