ISSN: 2278-4632
Vol-10 Issue-5 No. 9 May 2020

A New Approach for Java Based Kernel

Sakina Saifee[1], Dr. Sonali Kothari[2]
[1]Indore Institute of Science & Technology, Indore, Madhya Pradesh, India
[2]Sant Gadge Baba Amravati University, Amravati, Maharashtra, India

Abstract:

Kernel Interpreter is In-Kernel Interpreter that can make kernel understand the high level language. Kernel today only understands the system calls and work accordingly which makes it difficult for programmer to communicate with kernel. Here we introduce a new architecture for building in-kernel interpreter that will compiles high level language or Java code to native code or low level language instructions for execution in the kernel i.e. a kernel along with an interpreter that will correctly translate high-level language all its way to machine readable code, and will demonstrate that Java language can be integrated into kernel which could provide benefits with all its features to kernel and its process.

Keywords

High level Language Kernel, High Level Language OS, Java Based OS, Java Kernel, Kernel Interpreter Integration.

Introduction

Each computer system has a set of programs for user interaction and hardware interaction which is known as operating system or OS. Of all the programs in computer system most important program tends to be the kernel which is also called the heart of OS. Whenever system boots it has to be loaded into RAM(Random Access Memory) and also many other procedures that are critical are needed to be executed for the computer system to work. Whenever a program needs to use hardware resource it must first request the Operating System (OS). Then kernel is suppose to evaluate the request and then if the kernel decidesto give the permission to access the resource then it communicates with the right hardware component on the behalf of the user program. To include this mechanism today's operating system believes that the supply of specific hardware features that do not allow user programs for direct interaction with low-level hardware components or to grant access to arbitrary memory locations especially , the hardware that runs the minimum of two execution modes for the CPU differently i.e. for user programs a non privileged mode called User Mode and for the kernel a privileged mode called Kernel Mode.

Kernel

Kernels are the server that answers requests; these requests can come either from a process running on a CPU or an external device issuing an interrupt request. The Logic behind the kernel-user interface is upright: a user-space application get encode into the bytecode format and submit it to the kernel and then the kernel decodes the bytecode and again construct the filter. Then the kernel forwards the request to special programs called device drivers which control the hardware, manages the file system and sets interrupts for the CPU to enable multitasking. Many kernels also are responsible in checking that incorrect programs do not interfere with the operation of other programs, by denying access to memory that has never been allocated to them and restricting the total time CPU will consume. One of the most important advantage of having such kernel is that it will provide CPU scheduling, memory management, file management and other OS functions through system calls[1]. And also it's one large process running entirely during a single address space.

Issue with the kernels is the drivers. While installing any hardware into computer system one needs to make sure that the hardware has particular driver available. Now when you decide to switch your OS not all hardware might be compatible.

Also the major problem is the disconnection that occurs between users and kernel's developers[2]. Due to low level languages the user cannot operate or read the bugs without completely understanding the kernel.

Related Work

The J-Kernel [3] is an operating system built with Java Virtual Machine (JVM). It has been developed with the features of traditional operating system to provide the additional features that may not be found at the language level. J-Kernel goals to dene clear boundaries between protection domains. These are called task in J-Kernel. This makes management of resources and termination of task tractable and also analysis of the inner tasks communication get simplified. By limiting the shared types object among task it creates the boundary i.e., only special

ISSN: 2278-4632
Vol-10 Issue-5 No. 9 May 2020

object called capabilities can be shared and all other objects are directed towards the single task. This J-Kernel to efficiently revoke the capabilities without even adding any time cost to non-capability objects. The most significant benefit of J-Kernel is its flexible protection models, less time consumptions for software components communication and independency of OS. The Implementation of J-Kernel has been entirely in Java [4] and is expected to run on standard Java Virtual Machine (JVMs). The reason behind implementing Java was its capability for being used as general purpose safe language and also because of its virtual machine that are widespread and easy to work with. Java itself provides multiple protection domains by using single JVM which uses sandbox model for applet, and that model is at now very restrictive in use.

However, even with sophisticated optimization it seems likely that Java programs will not run as fast as C programs. Second, all current language-based protection systems are designed around a single language, which limits developers and doesn't handle legacy code. Software fault isolation [5] and verification of assembly language [6, 7, 8, 9] may someday offer solutions, but are still an active area of research.

Methodology Used for Existing Kernel

C Based Kernel

C language is nearly a portable programming language. It is the closest language to machine and also it is universally available for every processor's architecture. We can easily find out at least one C compiler for the every existing architecture. Nowadays we have highly optimized binaries that are been generated by modern compilers which makes it difficult to improve on their receiving output with hand written assembly codes. For system programming such as operating system and embedded systems C is the perfect language as it has arbitrary memory address access and also the pointer arithmetic features available. At the hardware or software boundary, computer systems and micro-controllers map their peripherals and Input/output pins into memory addresses. System's Applications should have to read and write to such custom memory addresses so that they can easily communicate with the planet. Hence C language's ability to manipulate with the arbitrary memory address is imperative for system programming. The matter arises because C features a very small run time. And therefore the memory footprint for its code is smaller than for many other languages. The crucial part of kernel's design is all about the abstraction levels that it provides where the security and policies should be implemented. Kernel security is very important mechanism for maintaining security at higher levels.

Java Based Kernel

Importance of Java in Kernel

Security may be a specialty for Java implementations generally. A java based OS benefits from language wide security measures to supply an honest set of security measures.

The Java programming language comprises of memory protection between threads and by completely eliminating pointers and enforcing strict checking on array access which restricts the access to the data members and member functions.

Java enables the class loader to load new classes into virtual machine at run time, these class loaders can be user defined also. This class loader will then fetch Java bytecode from some address or link of file system or URL and will submit the bytecode again to virtual machine. After this the Virtual machine will check on the bytecode for assuring it to be legal and will then integrate the new class. At the time of integrating if the bytecode will reference to other classes then class loader will be called recursively and will load those referring classes too.

Java Threads can implement suspend, setPriority and stop methods that modifies the state at which thread is upon. The affected domain can here call on other domains and then can suspend the thread so that the execution can be stopped while holding on to the critical lock or on the other resources. Conversely, the callee [1] that is malicious can hold on to a thread object and can modify its state after the return of an execution back to caller.

Java will make programs and driver's code portable and enable them to use the classpath facilities.

Java also can be liable for I/O operations, for instance by providing a call that permits the program to break into kernel mode to do port-based I/O.

Use of java libraries with the device drivers can resolve the matter of performances and code conversion. And also by using java the device drivers and system servers, can then be trusted, a big prerequisite in security aware sites.

Working of Java In Kernel Kernels implemented with Java are capable systems that may support multiple and cooperative tasks , which would run inside Java Virtual Machine (JVM) only. Such capabilities will help in access control list that is it will be implemented in naturally safe language, the principle of least privilege will be supported and also the execution of operations will quick.

A service provider on whom programs are going to depend on in order to function is not strictly necessary. All services and programs will be all part of a system in a way by which they can communicate with each other with simple function calls. Resource management will remain inherent part of the system.

Many security headaches appearing before will simply disappear by using the type safety and bounds checking features of the core Java language. The code that are been already executed will be ready for

ISSN: 2278-4632
Vol-10 Issue-5 No. 9 May 2020

dynamic optimization according to the load characteristics of the system and hence for the integrated architecture. Dynamic binary translation will ensure for legacy code execution.

Device drivers run in user mode and can use the Java libraries. JVM's multithreading facilities will be used implicitly which will give a positive impact on multiprocessor's performance which can also avoid the disturbance caused by Input / Output .

Implementation of separate domains for network and system servers and thus isolate the core system from a possible network security breach are often done.

The Java based Kernel [1] uses Java's built-in serialization features [10] to repeat an argument: the Java based Kernel will serialize an argument into an array of bytes, then deserialize the byte array to supply a fresh copy of the argument.
This makes it convenient as because the built in Java classes are serializable and it would involve overhead that is considerable. And therefore the Java based kernel will provide quick copy mechanism in which the direct copies of object with their fields would be made without any use of intermediary byte array.

Comparison Between Kernels

Table 1. Comparison between different kernels

System	Operation	Platform	Time
L4	Round trip IPC	P5-133	1.82µs
Exokernel	Protected control transfer(r/t)	Dec-5000	2.40µs
Eros	Round trip IPC	P5-120	4.90µs
J-Kernel	Method invocation with 3args	P5-133	3.77µs

The results are contrasted with a 3-argument method invocation in the Java based Kernel[1]. The Java based Kernel's performance is comparable with the three very fast systems. It is important to note that L4, Exokernel and Eros are implemented as a mix of C and assembly language code, while J-Kernel consists of Java classes without native code support. Improved implementations of JVMs and JITs are likely to enhance the performance of the J-Kernel[1].

Problem with Java Based Kernel

The Java based Kernel takes 60x to 180x longer than regular method invocation [1]. In MS-VM it takes significant fraction of cost in invocation which is necessary to enter the stub. It also needs the synchronization cost while changing thread segment and while looking up on current thread. Approximately these operations take for about 70% of the cross-task call on MS-VM and 80% on Sun-VM. Now the time required for NT kernel threads to switch between two contexts is 8.6s and here Java adds on to 1-2s of overhead. Here by this gives us confirmation about Java being costly on cross –task call situations for Java Based Kernel LRMI. Though the micro-benchmarks results are encouraging in which the time consumption of cross-task in Java based Kernel is 50x lower than in NT kernel. However it still incurs a stiff penalty over a plain method invocation. Critical code paths itself inspected that the management of threads and lock acquisition contributes to much of the time that is being consumed. The largest amount of time being used in cross-task is copying of the argument. The allocation of small objects also dominates the cost and appears difficult to optimize by more than a small factor.

Proposed Work

The canonical implementation of High Level language is an interpreter. A high level language is ideally an abstraction independent of particular implementations. The interpreter must analyze each statement within the program whenever it's executed then perform the specified action.

Here interpreters does not execute the source code as it is but convert it into machine likable form or some more compact internal form (Kernel recognizable form). Interpreter can also define high level language with which the Kernel Language(C Language) the semantics are given. It tells a reader about the expressiveness and elegance of OS. It also enables the interpreter to interpret its source code.

This Interpreter can present a highly customized user interface employing the user interface and input/output facilities of the language.

The most important and significant dimension of design and implementation of such interpreter is whether the feature of the high level language or interpreted language is been implemented with the equivalent feature in the low level language or the interpreter's hot language.

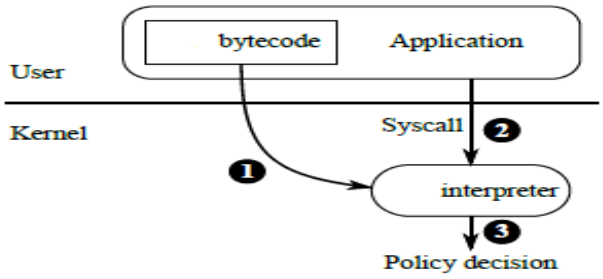

Figure 1. Presented flowchart for the Java interpreted kernel

Conclusion

This is a new approach for building in-kernel JIT interpreters that guarantee correctness through high-

ISSN: 2278-4632
Vol-10 Issue-5 No. 9 May 2020

level policy rules in user-space applications, to lower-level, across the user-kernel space boundary, and to native code in-kernel. It also guarantees advantages of using language-based protection are portability and good cross-domain performance. An analysis of known interpreter vulnerabilities demonstrates that it prevents all classes of security vulnerabilities for kernel interpreters. We believe that this is a promising direction since it achieves flexibility, safety, and good performance.

References

[i] Jitk: A Trustworthy In-Kernel Interpreter Infrastructure ,Xi Wang, David Lazar, Nickolai Zeldovich, Adam Chlipala, Zachary Tatlock MIT CSAIL and University of Washington

[ii] J-Kernel: a Capability-Based Operating System for Java Thorsten von Eicken, Chi-Chao Chang, Grzegorz Czajkowski, Chris Hawblitzel, Deyu Hu, and Dan Spoonhower ,Department of Computer Science Cornell University

[iii] C. Hawblitzel, C. C. Chang, G. Czajkowski, D. Hu, and T. von Eicken. Implementing Multiple Protection Domains in Java. 1998 USENIX Annual Technical Conference, p. 259-270, New Orleans, LA, June 1998.

[iv] J. Gosling, B. Joy, and G. Steele. The Java language specication.Addison-Wesley,1996.

[v] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Ecient Software-Based Fault Isolation. 14th ACMSymposium on Operating Systems Principles, p. 203216, Asheville, NC, December 1993.

[vi] G. Morrisett, D.Walker, K. Crary, and N. Glew. From System F to Typed Assembly Language. 25th ACM Symposium on Principles of Programming Languages. San Diego, CA, January 1998.

[vii] G. Necula and P. Lee. Safe Kernel Extensions Without Run-Time Checking. 2nd USENIX Symposium on Operating Systems Design and Implementation, p. 229243, Seattle, WA, October 1996.

[viii] G. Necula. Proof-carrying code. 24th ACM Symposium on Principles of Programming Languages, p. 106119, Paris, 1997.

[ix] Z. Shao. Typed Common Intermediate Format. 1997 USENIX Conference on Domain-Specific Languages, Santa Barbara, California, October 1997.

[x] JavaSoft. Remote Method Invocation Specification. http://java.sun.com.

