AT RUNTIME, INTERROGATE OR OTHERWISE ANALYZE APPLICATION PLATFORM TO SEE HOW DESIRED FUNCTIONALITY IS IMPLEMENTED

DOES ANALYSIS REVEAL THAT APPLICATION PLATFORM IS USING KNOWN FUNCTION THAT CAN BE MIMICKED?

USE TECHNIQUES TO MIMICK THE APPLICATION PLATFORM FUNCTIONALITY AND BYPASS HANDSHAKE

INITIATE THE HANDSHAKE AND USE KNOWN FUNCTION THROUGH APPLICATION PLATFORM

FIG. 3
Declarations under Rule 4.17:
— as to applicant’s entitlement to apply for and be granted a patent (Rule 4.17(H))
— as to the applicant’s entitlement to claim the priority of the earlier application (Rule 4.17(Hi))

Published:
— without international search report and to be republished upon receipt of that report
MIMICKING OF FUNCTIONALITY EXPOSED
THROUGH AN ABSTRACTION

BACKGROUND

[0001] In computing, an application platform refers to a framework, either in
hardware or software, which allows software to run. Typical platforms include a
computer's architecture, an operating system, or programming languages and their
runtime libraries. Thus, some examples of application platforms can include
operating systems such as MICROSOFT® WINDOWS® or Linux, or runtime
platforms such as MICROSOFT® .NET Framework and Java. It is common for an
application platform to have data or resources that are private to the application
platform. In such scenarios, if another application wants to use that same data, the
application first has to communicate with the application platform to request the
data, and then has to wait for the application platform to respond to the request.

[0002] An example of data that may be private to an application platform includes
thread local storage. Thread local storage is a technique by which an operating
system allows each thread to have its own private storage area for data. Access to
the data for a given thread that is stored in the private storage area has to be
performed through the operating system. This has a negative impact on the
performance of the application that is using the thread local storage.

SUMMARY

[0003] Various technologies and techniques are disclosed for mimicking
functionality of an application platform. At runtime, an analysis is performed of an
application platform to identify how a desired functionality is implemented by the
application platform. When the analysis reveals that the application platform is
using a known function that can be mimicked, a handshake with the application
platform is bypassed and the desired functionality is mimicked.

[0004] In one implementation, to mimic the functionality, a known function is
started for an application platform that uses the desired functionality. An entry
point of the known function is detected. Machine instructions are decoded in the
known function. Code that mimicks an effect of the machine instructions is
generated. The code that mimicks the effect of the machine instructions is then executed.

[0005] In one implementation, code that mimics an effect of machine instructions is generated by representing an effect of the decoded machine instructions as an abstract execution state. The decoded machine instructions are generated from the known function in the application platform that has the desired functionality that another application wishes to mimick. The abstract execution state is transformed into an abstract expression. Code is then generated from the abstract expression.

[0006] This Summary was provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] Figure 1 is a diagrammatic view of a typical handshake for accessing data or resources private to an application platform.

[0008] Figure 2 is a diagrammatic view of mimicking application platform functionality to get to data or resources that are otherwise private to an application platform.

[0009] Figure 3 is a process flow diagram for one implementation illustrating the high level stages involved in discovering and mimicking application platform functionality at runtime.

[0010] Figure 4 is a process flow diagram for one implementation illustrating the stages involved in mimicking the discovered functionality of the application platform.

[0011] Figure 5 is a process flow diagram for one implementation illustrating the stages involved in generating code that mimics the effect of the machine instructions.

[0012] Figure 6 is a process flow diagram for one implementation illustrating the stages involved in using mimicking in a scenario involving registers and register offsets.
DETAILED DESCRIPTION

The technologies and techniques herein may be described in the general context as an application that mimicks the functionality of an application platform, but the technologies and techniques also serve other purposes in addition to these. In one implementation, one or more of the techniques described herein can be implemented as features within any type of program or service that wants to access data or resources that are otherwise private to an application platform.

Figure 1 is a diagrammatic view of a typical handshake process for accessing data or resources private to an application platform. As mentioned previously, when an application wants to use data or resources that are private to an application platform, the application requesting data first has to communicate with the application platform to request the data or resources, and then has to wait for the application platform to respond to the request. Before the request can be processed, the application requesting data first has to establish a handshake with the application platform. The term "handshake" as used herein is meant to include a mechanism by which an application can request a service to be executed by an application platform on behalf of the application. The handshake is what opens a communication channel between the application requesting data and the application platform. Once the handshake is established, then the request can be submitted to the application platform, the application platform can access the data or resources to get the requested information, and return the data to the application requesting data.

In one implementation, as shown in Figure 2, a mimicking system is provided that allows application platform functionality to be mimicked so that data or resources can be accessed by an application that would otherwise be private to an application platform. In the diagram shown in Figure 2, the application requesting data uses a mimicking technique to directly access the data or resources that would otherwise have been private to the application platform. The term "mimick" as used herein is meant to include an execution of application code that...
has the same semantic effect as the application platform code that implements the service. The functionality of the application platform code being mimicked and the functionality of the application code are indistinguishable to an observer except for intended behavior changes such as better performance or added functionality.

[0017] One example where some or all of the mimicking techniques described herein can be used is with thread local storage. As noted earlier, thread local storage is a technique by which an operating system allows each thread to have its own private storage area for data. Access to the data for a given thread that is stored in the private storage area typically has to be performed through the operating system. Using some or all of the mimicking techniques described in further detail herein, the application 22 requesting data can directly access the data that is stored in thread local storage without having to go through the handshake and request process with the application platform (which would be the operating system in this example).

[0018] Turning now to Figures 3-6 with continued reference to Figure 2, some techniques for implementing mimicking system 20 are described in further detail. In some implementations, the processes of Figure 3-6 are at least partially implemented in the operating logic of computing device 500 (of Figure 7). Figure 3 is a process flow diagram 100 for one implementation illustrating the high level stages involved in discovering and mimicking application platform functionality at runtime. At runtime, an application platform is interrogated or otherwise analyzed to see how a particular functionality is implemented (stage 102). The particular functionality can be any functionality that is provided by an application platform that accesses private data or resources that another application now wants to access. If the analysis reveals that a known function is being used that can be mimicked (decision point 104), then techniques are used to mimic the application platform functionality and bypass the handshake (stage 106). The term "known function" as used herein is meant to include a function, procedure, routine, or other logic that can be executed separately from the application platform as a whole. The mimicking process is described in further detail in Figure 4.
[0020] If the analysis does not reveal that the application platform is using a known function that can be mimicked (decision point 104), then the handshake is initiated with the application platform and the known function (described in Figure 1) is used through the application platform to access the data or resources (as normal) (stage 108).

[0021] Turning now to Figure 4, a process flow diagram 120 for one implementation is shown that illustrates the more detailed stages involved in mimicking the discovered functionality of the application platform. At runtime, a known function is started that uses the desired functionality, such as an accessor function in the case of thread local storage (stage 122). The system determines the entry point of the function, which is typically the address of the first machine instruction (stage 124). The machine instructions are decoded (stage 126). If an alternate function has not already been generated for mimicking the effects of the desired functionality (decision point 128), then code is generated that mimicks the effect of the machine instructions, and the newly generated code is then executed (stage 132). This code generation process is described in further detail in Figure 5. If an alternate function has already been generated (decision point 128), then the existing code that has already been generated is executed to mimic the effect of the application platform (stage 132).

[0022] Figure 5 is a process flow diagram 150 that illustrates one implementation of the stages involved in generating code that mimics the effect of the machine instructions. The system represents the effect of the decoded machine instruction as an abstract execution state (stage 152). The term "abstract execution state" as used herein is meant to include facts about the program which independently reflect the state of the program at a given point in its execution. Facts needed to accurately represent program state include the semantic effects of instructions to be executed and the values of program data. A non-limiting example will now be discussed to further illustrate the concept of an abstract execution state. Suppose the interface specification of the application platform exposes a data accessor as a function called according to an Application Binary Interface or ABI. Suppose the ABI requires parameters such as the field offset to be written to memory (the
machine stack) and some registers to be saved before the call (to the machine stack) and restored after the call. The execution state can include the values of the machine registers and stack slots. The application emulates the instructions in the application platform but does not carry them out on actual values and registers/stack slots. Instead, variables are used as placeholders instead of actual values, and the abstract execution state maps registers/stack slots to expressions involving those variables (as opposed to the actual value that would result from evaluating the abstract expressions using concrete values for the variables).

[0023] The abstract execution state is optionally specialized with respect to the specific data field to be accessed (stage 154). For example, information exists which is available to an application program at runtime that is not available to the application platform when it is compiled. The code is specialized by making use of this runtime information to create more efficient code. The abstract execution state is then turned into an abstract expression (stage 156), and code is then generated from the abstract expression (stage 158). The term "abstract expression" as used herein is meant to include a representation of a program including the semantic effects of its instructions and the program data. A code generator can take an abstract expression as input and generate code which, when executed, evaluates the expression with the program's specific data values to produce a state which is an instance of the state described by the abstract expression.

[0024] Once code is generated from the abstract expression (stage 158), the code can optionally be Mined into the other code (stage 160). In-lining is a compiler optimization that can be performed to include lines of code directly in a program so that it is included as part of a compiled executable and thus faster than accessing the code from a separate library or executable. For example, the application that wants to mimick the functionality of the application platform can inline the code written to perform the mimicking directly with its other code to improve performance when executing the mimicked functionality. Figure 6 is a process flow diagram 180 that illustrates one implementation of the stages involved in using mimicking in a scenario involving registers and register offsets. The system determines which register and which offset the application platform is using (stage
182) and selects a method to use that will mimick the functionality (stage 184). The selected method is then executed using the base address and the register that was previously determined to be used by the application platform (stage 186).

[0025] A non-limiting example will now be described to further illustrate the concept of registers and register offsets. A given operating system (which is an application platform) may reserve some machine registers, which means that application code is not allowed to use them for their own storage purposes. These machine registers would be used by the operating system for bookkeeping purposes and accessed by operating system functions. In the example of thread-local storage, a given machine register may be initialized on thread creation to point to the "thread environment block". The thread environment block is a structure that contains, at an offset determined by the operating system, a field that points to an array of slots for thread-local storage. To mimick an access to thread local storage, a determination would have to be made regarding which reserved machine register points to the thread environment block and at what offset the thread local storage slots reside. For example, MICROSOFT® WINDOWS® NT on the x86 platform would use the FS register to point to the thread environment block and the offset would be 3600+4*(thread local storage slot index) for the first 64 thread local storage slots. In the process described in Figure 6, this base address and the FS register would be used to execute the selected method (stage 186).

[0026] Turning now to Figure 7, an exemplary computer system to use for implementing one or more parts of the system is shown that includes a computing device, such as computing device 500. In its most basic configuration, computing device 500 typically includes at least one processing unit 502 and memory 504. Depending on the exact configuration and type of computing device, memory 504 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. This most basic configuration is illustrated in Figure 7 by dashed line 506.

[0027] Additionally, device 500 may also have additional features/functionality. For example, device 500 may also include additional storage (removable and/or non-removable) including, but not limited to, magnetic or optical disks or tape.
Such additional storage is illustrated in Figure 7 by removable storage 508 and non-removable storage 510. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Memory 504, removable storage 508 and non-removable storage 510 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by device 500. Any such computer storage media may be part of device 500.

[0028] Computing device 500 includes one or more communication connections 514 that allow computing device 500 to communicate with other computers/applications 515. Device 500 may also have input device(s) 512 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 511 such as a display, speakers, printer, etc. may also be included. These devices are well known in the art and need not be discussed at length here.

[0029] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims. All equivalents, changes, and modifications that come within the spirit of the implementations as described herein and/or by the following claims are desired to be protected.

[0030] For example, a person of ordinary skill in the computer software art will recognize that the examples discussed herein could be organized differently on one or more computers to include fewer or additional options or features than as portrayed in the examples.
CLAIMS

What is claimed is:

1. A computer-readable medium having computer-executable instructions for causing a computer to perform steps comprising:
   - at runtime, performing an analysis of an application platform to identify how a desired functionality is implemented by the application platform (102); and
   - when the analysis reveals that the application platform is using a known function that can be mimicked, then bypassing a handshake with the application platform and mimicking the desired functionality (106).

2. The computer-readable medium of claim 1, further having computer-executable instructions for causing a computer to perform steps comprising:
   - when the analysis reveals that the application platform is not using a known function that can be mimicked, then initiating the handshake with the application platform and using the desired functionality through the application platform (108).

3. The computer-readable medium of claim 1, wherein the mimicking of the desired functionality step comprises the steps of:
   - starting the known function of the application platform (122);
   - determining an entry point of the known function (124);
   - decoding machine instructions in the known function (126);
   - generating code that mimicks an effect of the machine instructions (130);
   - and executing the code that mimicks the effect of the machine instructions (130).

4. The computer-readable medium of claim 1, wherein the decoding the machine instructions step comprises the steps of:
   - representing an effect of decoded machine instructions as an abstract execution state (152); and
   - transforming the abstract execution state into an abstract expression (156).

5. The computer-readable medium of claim 4, wherein the generating code step is operable to generate the code from the abstract expression (158).
6. The computer-readable medium of claim 1, wherein the mimicking the desired functionality step comprises the steps of:
   identifying a register and a register offset that the application platform is using to implement the desired functionality (182); and
   executing a method designed to mimick the desired functionality by using a base address and the register that was identified (186).
7. The computer-readable medium of claim 6, further having computer-executable instructions for causing a computer to perform steps comprising:
   selecting the method designed to mimick the desired functionality from a plurality of available methods (184).
8. The computer-readable medium of claim 1, wherein the desired functionality is thread local storage (184).
9. The computer-readable medium of claim 8, wherein the known function is an accessor function (184).
10. A method for mimicking functionality of an application platform at runtime comprising the steps of:
   at runtime, starting a known function of an application platform that uses a desired functionality (122);
   determining an entry point of the known function (124);
   decoding machine instructions in the known function (126);
   generating code that mimicks an effect of the machine instructions (130);
   and
   executing the code that mimicks the effect of the machine instructions (130).
11. The method of claim 10, wherein the generating step is performed when the code has not previously been generated to mimick the effect of the machine instructions (130).
12. The method of claim 10, wherein the desired functionality is thread local storage (122).
13. The method of claim 12, wherein the known function is an accessor function (122).
14. The method of claim 10, wherein the entry point of the known function is determined as a first machine instruction of the machine instructions in the known function (124).

15. The method of claim 10, further comprising:

prior to executing the code, inlining the code that mimicks the effect with other code to achieve a faster execution performance (160).

16. A method for generating code that mimics an effect of machine instructions comprising the steps of:

representing an effect of decoded machine instructions as an abstract execution state, the decoded machine instructions having been generated from a function in an application platform that has a desired functionality that another application desires to mimick (152);

transforming the abstract execution state into an abstract expression (156); and

generating code from the abstract expression (158).

17. The method of claim 16, further comprising:

prior to transforming the abstract execution state, specializing the abstract execution state to a specific data field to be accessed (154).

18. The method of claim 16, further comprising:

inlining the generated code into other code (160).

19. The method of claim 18, wherein the other code is Mined directly into the other application that is mimicking the desired functionality (160).

20. The method of claim 19, wherein the other code is Mined directly into the other application to improve a performance of the other application when mimicking the desired functionality (160).
AT RUNTIME, INTERROGATE OR OTHERWISE ANALYZE APPLICATION PLATFORM TO SEE HOW DESIRED FUNCTIONALITY IS IMPLEMENTED

102

DOES ANALYSIS REVEAL THAT APPLICATION PLATFORM IS USING KNOWN FUNCTION THAT CAN BE MIMICKED?

104

YES

USE TECHNIQUES TO MIMICK THE APPLICATION PLATFORM FUNCTIONALITY AND BYPASS HANDSHAKE

106

NO

INITIATE THE HANDSHAKE AND USE KNOWN FUNCTION THROUGH APPLICATION PLATFORM

108

FIG. 3
120

AT RUNTIME, START A KNOWN FUNCTION THAT USES DESIRED FUNCTIONALITY

122

DETERMINE ENTRY POINT OF FUNCTION

124

DECODE MACHINE INSTRUCTIONS

126

ALTERNATE ALREADY AVAILABLE?

128

YES

NO

GENERATE CODE THAT MIMICS THE EFFECT OF THE MACHINE INSTRUCTIONS AND EXECUTE THE CODE

130

EXECUTE THE CODE THAT MIMICS THE EFFECT OF THE APPLICATION PLATFORM

132

FIG. 4
FIG. 5

150

REPRESENT EFFECT OF DECODED MACHINE INSTRUCTIONS AS ABSTRACT EXECUTION STATE

152

OPTIONALLY SPECIALIZE ABSTRACT EXECUTION STATE WITH RESPECT TO SPECIFIC DATA FIELD TO BE ACCESSED

154

TURN ABSTRACT EXECUTION STATE INTO ABSTRACT EXPRESSION

156

GENERATE CODE FROM ABSTRACT EXPRESSION

158

OPTIONALLY INLINE GENERATED CODE INTO OTHER CODE

160
180

DETERMINE WHICH REGISTER AND WHICH OFFSET THE APPLICATION PLATFORM IS USING

182

SELECT METHOD TO USE THAT WILL MIMICK THE FUNCTIONALITY

184

EXECUTE SELECTED METHOD USING THE BASE ADDRESS AND REGISTER PREVIOUSLY DETERMINED

186

FIG. 6