Transforming SQLITE to Run on a Bare PC

Uzo Okafor, Ramesh K. Karne, Alexander L. Wijesinha and Bharat S. Rawal
Department of Computer and Information Sciences, Towson University, Towson, Maryland 21252 U.S.A.

Keywords: Code Transformation, Bare PC Systems, Bare Machine Computing, SQLITE, Database Systems.

Abstract: SQLITE is a popular small open-source database management system with many versions that run on popular platforms. However, there is currently no version of the SQLITE application that runs on a bare PC. Since a bare PC does not provide any form of operating system (or kernel) support, bare PC applications need to be completely self-contained with their own interfaces to the hardware. Such applications are characterized by small code size, and have inherent security and performance advantages due to the absence of a conventional operating system. We describe a general transformation approach that can be used to transform the SQLITE application to a lean SQLITE application that runs on a bare PC. We present the current state of this work and identify several important issues that need further research.

1 MOTIVATION

We have developed a variety of bare PC applications (Appiah-Kubi, 2009 and He, 2008) that run on a machine without using any operating system (OS), or kernel. Bare PC systems are also different from embedded systems. The bare PC architecture is simple, application-centric, extensible, and lean and independent of any operating environment. At present, it is based on a single programming language C++ with some low-level interface code written in C or assembly language. The SQLITE application is a system-level program with reasonable complexity. It is written in C. The developers of this application created an amalgamation package, delivered as two source files with all environment parameters included in them. This application is suitable for transformation to a bare PC, as it is complex, and provides insight for developing an automated tool for transforming other applications.

2 INTRODUCTION

The SQLITE application in amalgamation form for Windows OS (Operating System) has two source files (shell.c, sqlite3.c) and two header files (sqlite3.h, sqlite3ext.h). The sizes of shell.c and sqlite3.c are 86, 016 and 4,323,826 bytes respectively. The total number of lines of code in both source files is 129,003, of which 55,691 are commented lines of code and 40,297 are executable statements. The code runs in the Microsoft Visual Studio 2010 Express environment. When the application runs, a user can perform standard database functions including those to create tables, insert data and query a database. The output will be displayed in a Microsoft Window (there is no graphics interface). Our task is to transform this application code so that it will run on a bare PC without using any operating system/kernel or embedded software.

This transformation process is different from the translation done by compilers, interpreters, programming language parsers and other tools used for porting. The primary issue involved in transforming conventional applications to bare applications is that a bare application needs to run as a completely self-supporting entity. Transforming the code involves eliminating operating system dependencies and system calls from the code, and using a direct hardware API that is based on the Bare Machine Computing (BMC) paradigm (Karne 2005 and Khaksari 2012). This paradigm supports the development of lean, minimal and low cost applications as proposed in (Soumya, Guerin and Hosanagar 2011).

There is a considerable amount of work relating to code transformation and translation e.g., Intel x86 programs can be translated from binary code to

Okafor U., K. Karne R., L. Wijesinha A. and S. Rawal B..
Transforming SQLITE to Run on a Bare PC.
DOI: 10.5220/0004015403110314
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 311-314
ISBN: 978-989-8565-19-8

ARM and Alfa binaries with reasonable code densities and quality (Hwoang, Lin and Chang, 2010). In (Schoeberl, Korsholm, Kalibera and Ravn, 2011) the Java virtual machine is implemented directly on hardware in an embedded system (as extensions of standard interpreters and hardware objects, which interface directly with the JVM). SoulPad presents a new approach based on carrying an auto-configuring operating system along with a suspended virtual machine on a small portable device. With this approach, the computer boots from the device and creates a virtual machine, thus giving users access to their personal environment, including previously running computations. BulkCompiler is a simple compiler layer that works with group-committing hardware to provide a whole-system high-performance sequential consistency platform. They introduce ISA primitives and software algorithms to drive instruction-group formation, and to transform code to exploit the groups. None of these approaches provide a technique to transform OS-based code to run on a bare PC or a bare machine.

The rest of the paper is as follows. Section 3 describes the BMC paradigm and its characteristics. The transformation process is described in Section 4. Some transformation process issues are identified in Section 5. Section 6 contains the conclusion.

3 BMC AND RELATED WORK

A conventional OS/kernel or embedded software acts as middleware between the hardware and an application. An application programmer is therefore isolated from an application's execution environment and plays a very limited role in resource control and management. Thus, the programmer has no direct control of program execution, which is managed by the OS. In contrast, Bare Machine Computing (BMC), also called dispersed operating system computing (DOSC), uses the BMC paradigm, which eliminates the OS by introducing an application object (AO). An AO allows the programmer to assume sole control of the application and its execution. Therefore, execution and control knowledge of a given BMC application now resides with the AO programmer. The BMC paradigm differs from conventional computing in two major ways. First, the machine itself is bare with no existing software i.e., no resources such as the OS or applications reside on the machine and need to be protected. Second, an AO programmer is totally responsible for program execution, control and management. When a device is bare and has no valuable resources such as a hard disk, OS or a kernel to protect, it only becomes necessary to secure the AO and its hardware interfaces. In the BMC approach, mass storage can also be on a network. An AO is built for a given set of applications to run on a bare machine as a single monolithic executable that is lean and has minimal (only essential) functionality. The necessary network protocols are intertwined with the application and are not part of the OS-controlled stack or kernel i.e., there is no notion of user space and kernel space as in a conventional system. The code for boot, load, executable, data and files are stored on an external storage device such as a USB. When this device is plugged-in, the machine boots and runs its own program without using any external programs or software. No dynamic link libraries or virtual machine code are present or allowed in BMC. When an AO is running in the machine, only the software loaded by the user will run. BMC systems are currently being investigated for their security benefits.

In the bare machine computing (BMC) paradigm, a software program is designed as an AO, which is self-supporting, self-managed and self-executable. Once an AO is booted and loaded into a bare PC, it runs without using any other software i.e., there is no centralized support of any kind. The BMC paradigm has been used to develop complex applications such as Web servers, Webmail servers, and VoIP soft-phones. These applications use lean versions of standard network protocols, and security protocols such as TLS, if needed. The BMC approach completely eliminates the underlying OS and centralized middleware. While the BMC paradigm is similar to approaches that reduce OS overhead and/or use lean kernels such as Exokernel (Engler,1998), IO-Lite (Pai, Druschel and Zwaenepoel, 2000) and Palacios and Kitten (Lang, et al ,2010) there are significant differences. In particular, there is no centralized code that manages system resources i.e., the application is completely self-supporting, and the AO programmer controls and manages the execution environment and the hardware. The bare PC application development process and BMC characteristics are described in greater detail in (Khaksari, Karne and Wijesinha, 2012).

4 TRANSFORMATION PROCESS

We now describe the transformation process in

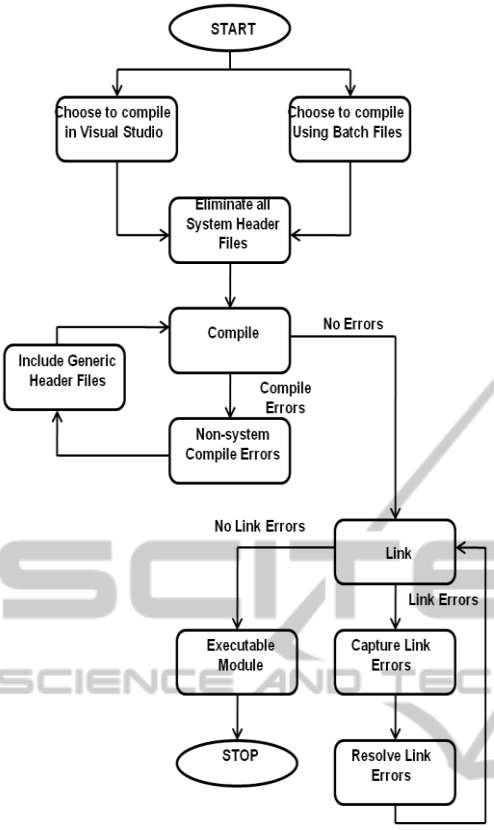

Figure 1: Transformation Process.

detail. Although our focus here is on transforming SQLITE to run on a bare PC, the techniques and underlying principles for transforming other applications will be similar. The overall methodology for transformation is illustrated in Fig. 1. The SQLITE application can be compiled using a batch execution file like cpp.bat in a DOS environment, or it can be compiled using the Visual Studio environment. When the latter environment is chosen, it is necessary to set the options to create a bare executable. When a batch file option is used, the application is compiled in a DOS window environment. In both cases, the ultimate objective is to eliminate any OS dependencies that may be due to system header files. Removal of a pure system header file will not cause any compile-time errors. However, removal of a non-system header file will generate compiler errors since the source code may be using some standard definition of variables or structures. To resolve errors, it may be necessary to use some new declarations for variables and functions, and/or to insert some generic header files. The compilation process is iterative until all compiler errors are resolved.

The next step is linking. When the above piled objects are linked, link errors show up mostly as system calls or OS-dependent interfaces. Each OS environment has its own system calls. System calls can be classified based on their type. As with compiling, the linking process is also iterative until all link errors are resolved. Link errors can be resolved by implementing a system call in a bare PC environment. If the bare PC implementation was not yet done, we simply put in a prototype and postponed the implementation. Once all link errors are resolved, we can create an executable that is ready to be loaded in a bare PC.

Compiling and linking as described above are currently done manually. In future, it will be possible to develop an automated tool for these steps (ideally, by modifying a Visual Studio tool).

The SQLITE linking process resulted in 88 link errors. In general, we expect a large number of link errors. The transformation process has to resolve these errors in order to generate an executable that can run on a bare PC.

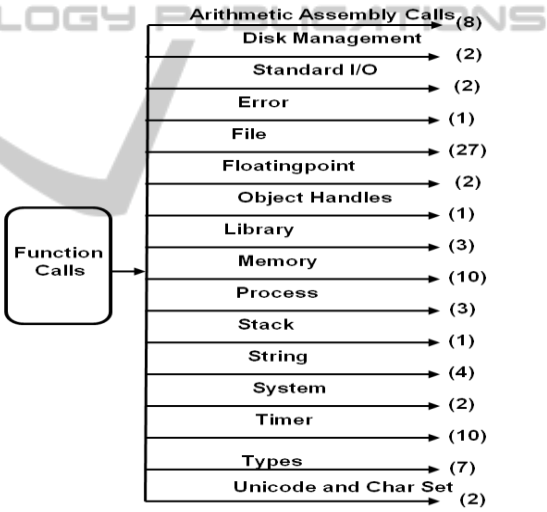

Figure 2: Classification of System Calls.

The 88 link errors are due to different types of system calls: we categorize them as shown in Fig. 2. Some of these calls are duplicates as they are referenced in two different source files. Some functions such as string type functions are not related to the underlying OS. However, many such functions and the most commonly used application-level functions are typically bundled with OS libraries. This makes these applications OS dependent and makes it harder to port them to other platforms. The nature of system calls varies according to the particular OS environment in use. For example, 542 system calls are cross-referenced for Linux systems in (FreeBSD/Linux). In our

application, we do not need to handle all system calls that are available in a commercial OS. The above system calls are implemented in BMC as direct hardware interfaces (Karne, Jaganathan and Ahmed, 2005) which are controlled by an AO programmer

5 TRANSFORMATION ISSUES

In theory, the transformation process is complete once all system header files are removed and the necessary bare PC interfaces are provided. However, this proved to be not the case in practice. The major obstacles encountered and other problems include: executable formats, memory map and layout, base address for relocation, and memory allocation. The major items in transforming the SQLITE application to a bare PC application can be summarized as follows:

- It is a large and complex application that also interacts with a database
- It uses its own mini-OS structures
- It is very much intertwined with OS header files and system calls
- It uses internal cache, paging, and memory management
- It built its own I/O calls on top of standard I/O.

The issues identified above are unique to the SQLITE program. Similar issues will arise and need to be addressed when transforming complex applications that are closely integrated with the underlying OS and environments.

6 CONCLUSIONS

Transforming complex applications to run on a bare PC with no operating system or kernel support is not trivial. We discussed a methodology for transforming SQLITE and detailed the steps for compiling, linking and running the application on a bare PC. We also identified several design issues related to the transformation process. In particular, we identified system calls related to the SQLITE application and classified them into groups. The direct bare hardware interfaces (hardware APIs) are general and can be used to build any bare PC application (not just SQLITE). This work serves as a foundation for transforming conventional OS-based applications to run on a bare PC and will help in building an automated transformation tool in the future.

REFERENCES

Ahn, W., Qi, S., Nicolaides, M., and Torrellas, J., 2009. BulkCompiler: High-Performance Sequential Consistency through Cooperative Compiler and Hardware Support. In IEEE/ACM International Symposium on Micro Architecture.

Appiah-Kubi, P., Karne, R., and Wijesinha, A., 2010. The Design and Performance of a Bare PC Webmail Server. In the 12th IEEE International Conference on High Performance Computing and Communications, AHPCC.

Engler, D., 1998. The Exokernel Operating System Architecture. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Ph.D. Dissertation.

FreeBSD/Linux Kernel Cross Reference, http:// fxr.watson.org/ fxr/source/kern/syscalls.c.

He, L., Karne, R., and Wijesinha, A., 2008. Design and Performance of a bare PC Web Server. In International Journal of Computer and Applications, Acta Press.

Hwang,Y., Lin,T., Chang, R., 2010. DisIRer: Converting a Retargetable Compiler into a Multiplatform Binary Translator. In ACM Transactions on Architecture and Code Optimization , Vol. 7, Issue 4.

Karne, R., Jaganathan,K., Ahmed, T., and Rosa, N., 2005. DOSC: Dispersed Operating System Computing. In 20th Annual ACM Conference on Object Oriented Programming (OOPSLA).

Karne, R., Jaganathan,K., and Ahmed, T., 2005. How to run C++ Applications on a bare PC. In 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel Distributed Computing (SNPD), p. 50-55.

Khaksari, G., Karne, R., and Wijesinha, A., 2012. A Bare Machine Application Development Methodology. In International Journal of Computer Applications (IJCA), Vol. 19, No.1.

Lange, J., Pedretti, K., Hudson,T., Dinda, P., Cui, Z., Xia, L., Bridges, P., Gocke, A., Jaconette, S., Levenhagen, M., and Brightwell, R., 2010. Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing. In Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium.

Pai, V., Druschel, P., and Zwaenepoel, W., 2000. IO-Lite: A Unified I/O Buffering and Caching System. In ACM Transactions on Computer Systems, Vol.18 (1), p. 37-66, ACM.

Schoeberl, M., Korsholm, K., Kalibera, T., and Ravn, A., 2011. A Hardware Abstraction Layer in Java. In ACM Transactions on Embedded Computing Systems, Vol.10, No.04, Article 42.

Soumya,S., Guerin, R., and Hosanagar, K ., 2011. Functionality-rich Vs Minimalist Platforms: A Two-sided Market Analysis. In ACM Computer Communication Review, Volume 41, Number 5, p36-43.