

TOWSON UNIVERSITY
OFFICE OF GRADUATE STUDIES
TRANSFORMING SQLITE DBMS TO RUN ON A BARE PC

by

Uzo Okafor

A Dissertation

Presented to the faculty of

Towson University

in partial fulfillment of

requirements for the degree of

Doctor of Science in Information Technology

Department of Computer and Information Sciences

Towson University

Towson, Maryland 21252

May 2013

© 2013 By Uzo Okafor

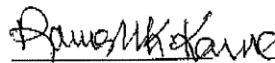
All Rights Reserved

TOWSON UNIVERSITY
COLLEGE OF GRADUATE STUDIES AND RESEARCH

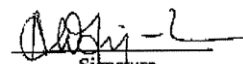
DISSERTATION APPROVAL PAGE

This is to certify that the dissertation prepared by Uzo Okafor entitled "TRANSFORMING SQLITE DBMS TO RUN ON A BARE PC" has been approved by this committee as satisfactory completion of the requirement for the degree of Doctor of Science in Information Technology.

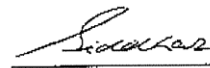
Dr. Ramesh K. Karne
Chair, Dissertation Committee

 4-26-2013
Signature Date

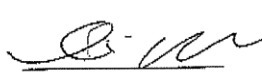
Dr. Alexander L. Wijesinha
Committee Member

 4/26/13
Signature Date


Dr. Siddharth Kaza
Committee Member

 4/26/13
Signature Date

Dr. Wei Yu
Committee Member

 4/26/2013
Signature Date

Janet Delany
Dean of Graduate Studies

 5/6/13
Signature Date

ACKNOWLEDGEMENTS

I would like to express my appreciation to all those who have supported my efforts to complete this dissertation. I am greatly appreciative of my research committee Dr. Ramesh K. Karne (chair), Dr. Alexander Wijesinha, Dr. Siddharth Kaza and Dr. Wei Yu for supporting this research. I am especially thankful to Dr. Karne and Dr. Wijesinha for all the long hours in lab, the support and advice I have received throughout this dissertation research. I am also thankful to my elderly mother, Dorah Okafor, for continuing to be with us – as it were, patiently. I have to thank my wife, Gladiette Nkiru, my children - especially Chika Jennifer Okafor who just passed her New York Law Bar exam, as it were, to make our successes complete - and several friends and relatives who have passionately supported my academic goals during the long period of my doctoral study.

I also would like to thank Dr. Chao Lu, Chair of the Department of Computer and Information Sciences and Towson University for facilitating this work. My gratitude also goes to the late Frank Anger (National Science Foundation) for his support of the Application oriented Object Architecture, which evolved into Bare Machine Computing research which in turn led to this dissertation.

ABSTRACT

TRANSFORMING SQLITE DBMS TO RUN ON A BARE PC

Uzo Okafor

This dissertation extends on-going Bare Machine Computing (BMC) research at Towson University. BMC applications run on a bare machine without any commercial operating system, kernel or other centralized support and are in need of a bare machine database management system. This research deals with transforming SQLITE DBMS system to run on a bare PC. The SQLITE DBMS is a lean database that runs on Windows or Linux operating system. It is commonly used as a standalone database across many academic institutions and is also a free download. When a DBMS runs on top of an operating system, it uses the operating system's system calls to access hardware resources. This dissertation will eliminate such system calls and allow the SQLITE DBMS to directly access and manage hardware resources. This novel concept eventually will pave the way to transform other systems and application programs.

The transformation process poses many daunting challenges and issues. There are a variety of ways to explore the transformation process. However, this dissertation proposes a novel approach in transformation methodology using existing tools. It uses Microsoft Visual Studio to develop, test, validate and debug bare PC applications; this step is referred to as pseudo transformation process. This pseudo transformed code is then used in further transformation process where rest of the OS related dependencies are eliminated. This fully transformed code is now ready to run on a bare PC.

The primary objective of this research assumes minimal understanding or modification of the `SQLITE` code during its transformation. This dissertation demonstrates and validates this hypothesis successfully. After transformation, the functionality and results are validated with its original `SQLITE` model that runs on a Windows operating system. This investigation serves as a cornerstone for future transformation of operating system based applications to run on a bare PC or a bare machine. It also lays a foundation to build an automated tool to replace the manual process outlined in this research.

TABLE OF CONTENTS

TABLE OF CONTENTS.....	vii
LIST OF FIGURES.....	x
Chapter1 _MOTIVATION.....	1
Chapter 2 INTRODUCTION.....	2
Chapter 3 THE PROBLEM AND THE HYPOTHESIS.....	5
3.1 The Problem.....	5
3.2 Hypothesis	6
3.3 Approach.....	6
Chapter 4 SQLITE DBMS STRUCTURE AND CHARACTERISTICS.....	8
Chapter 5 BARE MACHINE COMPUTING AND INTERFACES.....	9
5.1 BMC Computing Paradigm	9
5.2 BMC Application Development.	11
5.3 Direct Hardware Interfaces.....	14
5.3.1 Static and Dynamic Memory.....	14
5.3.2 User Interfaces.....	15
5.3.3 Network Interfaces.....	15
5.3.4 Process Interfaces.....	16
5.3.5 File Interfaces.....	17
5.3.6 Boot and Load Interfaces.....	17
5.3.7 Compile, Link and Library Issues.....	17

5.4	Bootable USB	18
5.5	Memory Map.	20
Chapter 6 TRANSFORMATION STRATEGIES.....		23
6.1	Same Executable.....	24
6.2	Trap System Calls	25
6.3	Resolve at Assembly Level.....	26
6.4	Remove all Header Files	27
Chapter 7 TRANSFORMATION PROCESS.....		29
7.1	Scaling Down Features	31
7.2	Visual Studio Application.....	31
7.3	Bare PC Application	36
Chapter 8 RESULTS AND DISCUSSION.....		41
Chapter 9 RELATED WORK.....		48
Chapter 10 SIGNIFICANT CONTRIBUTIONS.....		51
Chapter 11 SUMMARY.....		53
APPENDICES.....		54
APPENDIX A: SQLITE RESOURCES.....		55
APPENDIX B: THE TWO DEVELOPMENT PLATFORMS.....		56
REFERENCES.....		68
CURRICULUM VITA.....		72

LIST OF ABBREVIATIONS

OS	Operating Systems
VS	Visual Studio
BMC	Bare Machine Computing
PC	Program Counter
(bare) PC	bare Personal Computer
API	Application Programming Interface
DBMS	Data Base Management System

LIST OF FIGURES

Figure 1 Conventional OS and Bare Machine Computing.....	25
Figure 2 Steps in Developing Bare Machine	27
Figure 3 USB Layout	33
Figure 4. Memory Map	35
Figure 5 Transformation Strategies.....	41
Figure 6 Transformation Methodology	44
Figure 7 Visual Studio (VS) application platform.....	47
Figure 8 OS related Calls and Library Functions	49
Figure 9 Bare PC application platform.....	52
Figure 10 Batch Files	52
Figure 11 Simple Queries QQ1 on bare PC	57
Figure 12 Matching results on Visual Studio for QQ1	57
Figure 13 Error E1 flagged by Parser on bare PC	58
Figure 14 same error E1 on Visual Studio - see Fig 13.....	58
Figure 15 More Simple Queries on bare PC	59
Figure 16 same queries on Visual Studio as in Fig 15.....	59
Figure 17 Bare PC Output Display of more complex queries QQ2	60
Figure 18 same complex queries QQ2 on Visual Studio.....	60
Figure 19 asm.bat file.....	66
Figure 20 cpp.bat file.....	67
Figure 21 ln.bat file.....	68
Figure 22 mk.bat file.....	69
Figure 23 System Calls from using NODEFAULTLIB option.....	71
Figure 24 bare PC memory object - Initialization.....	72
Figure 25 bare PC memory object – Allocate.....	73
Figure 26 bare PC memory object – Allocate Continued.....	74
Figure 27 bare PC memory object – free memory.....	75
Figure 28 bare PC memory object – Reallocate.....	76
Figure 29 bare PC memory object – Reallocate Continued.....	77

Chapter 1 MOTIVATION

A variety of bare PC applications were developed at Towson University [3, 7, 10, 17] that run on a bare PC without using any operating system (OS), kernel, or any embedded system. These applications need a database management system that runs on a bare PC. The bare PC architecture is simple, application-centric, extensible, lean and independent of any operating environment. At present, it is based on a single programming language C/C++ with some low-level interface code written in C or assembly language. The SQLITE database management system (DBMS) [26] application is a system-level program with reasonable complexity and 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. This application also does not involve graphics or networking, which enables us to focus on transforming the application itself.

Chapter 2 INTRODUCTION

Application programs written in a high level programming language are translated to machine code by a compiler based on underlying machine architecture and operating system environment. Each program also needs system calls/libraries to access hardware resources. An operating system acts as a middleware to provide hardware abstractions to application programs. Thus, application programs are not truly independent of its execution environment. When application programs are made totally independent of execution and operating environments, they become portable across many pervasive devices. This is a different way of achieving ubiquity without using a virtual machine. When applications are made to run on a bare machine, they are not susceptible to rapid changes in operating systems and computing platforms. This will bring a revolution in computing where application programs are polarized on applications instead of computing environments and platforms. This is not same as ubiquity paradigm provided by Java or other virtual machines as they are not bare machine systems!

In essence, an application program is intertwined with OS and also its underlying machine architecture. Ideally, application programs should be independent of OS and also machine architecture to make software reusable and portable across pervasive devices. The system calls/libraries inserted by the compiler are provided by an underlying OS in addition to other hardware abstractions. The high level language translation to machine code is dependent upon the machine architecture. This dissertation focuses on eliminating hardware abstractions (OS or middleware) by providing direct hardware interfaces to application programs [15]. A unified compiler approach is needed to

generate machine code for different machine architectures, which is not part of this dissertation.

The transformation process in the beginning stages explored several methodologies and avenues. Many of these methodologies ended in road blocks in the transformation process. In one approach, where we start the transformation process in a bare PC, there is no working code in the beginning of transformation. There are no development environments (e.g. Visual Studio for bare PC applications) for developing, testing, validating and debugging the code. When the code does not work, it is hard to determine whether the problem is in the SQLITE or in our transformation. In a bare PC, system calls/libraries are replaced with direct hardware APIs so that applications can directly communicate with hardware. These APIs were not tested before for transformation process.

Another approach experimented with, was where you go through step-by-step testing of the code in a bare PC and compare with the same steps in the original code running on Visual Studio IDE. This was very tedious and time consuming process due to the complexity of SQLITE DBMS code and its internal structures. As example, in some instances, there were several thousand lines of code within a single switch statement and all the cases associated with it. This approach fell apart quickly because it was tedious and frustrating.

The SQLITE DBMS code size also caused us problems on the bare PC platform; the search for the needed solutions on bare PC forced us into re-engineering the bare PC platform several times. The code size being larger than 64K, old Visual C++ compilers cannot compile code of this size. The new C/C++ compiler obtained from Visual Studio

8.0 also caused many issues with executable layouts. In this exe model layout, code and data are not contiguous – we discovered that there is a gap between code and data. The starting of the data after code was on a page boundary (4K or 0x1000). The loader should take this into consideration while loading code and data.

The frustration from the transformation process almost forced an end to our effort on this research many times. As we progressed, another problem for transformation was identified to be the lack of a testing and validation tool. The Visual Studio tool is already available for testing OS based applications and was not meant for testing on a platform without an OS. And then, we discovered how to test a bare PC application using Visual Studio. To use the same (Visual Studio) tool for bare PC development that is used normally for applications running in a regular OS-environment turned out to be the right approach to make the transformation easier and faster. However, it is not possible to completely transform an application to bare PC application on Visual Studio as it uses many hardware resources from its underlying OS but discovering how to use Visual Studio for any portion of the total transformation task was still a game-changing event. It was a significant achievement that gets us off the ground delivering to us, an application that was almost completely transformed; the discovery of this pseudo-transformation approach indeed gave us momentum in the right direction.

The above introduction is necessary and essential to understand the research process and the stumbling blocks in the process. The rest of the sections in this dissertation provide more details of theories and methodologies of application transformation, the transformation process itself, the issues and results.

Chapter 3 THE PROBLEM AND THE HYPOTHESIS

3.1 The Problem

In BMC laboratory at Towson University, many BMC applications have been developed from scratch, which have demonstrated significant improvements in performance over conventional applications that run on an OS platform. Up to this point, we do not have a DBMS that runs on the bare PC platform. To have a complimenting DBMS running on the same (bare PC) platform spells overall advancement and progress in this environment. If we can develop this DBMS, it would be the first of such a badly needed application in our bare environment; we would have blazed a trail. Any one of the applications that have already been completed in bare PC environment would now have a database that it may use.

The approach of starting from scratch requires intimate understanding and expertise in the BMC programming paradigm. It does not – but should – take advantage of existing applications and freely available source code and resources for these applications. We believe it is much less work, time and cost – if an existing application with all of its functionalities is transformed to run in our environment.

The BMC paradigm is behind all the work that has already been completed on bare PC processing. Not surprisingly, the (BMC) paradigm continues to drive for even more device drivers that do not use code related to an OS platform or any OS related calls. Developing such BMC drivers has been a daunting obstacle causing a shortage of BMC drivers. This in turn contributes to the difficulty in making the BMC applications popular and marketable in the world. One solution to the scarcity in such BMC drivers is to

transform existing drivers which now run in an OS environment to make them run in a bare PC environment.

3.2 Hypothesis

The hypothesis of this dissertation is as follows. Most of the computer applications today run on top of the existing operating system and require either system calls or some sort of API to reach the hardware. A Bare Machine Computer application runs directly on top of hardware without any need for a centralized operating system or a kernel. But, it has its own hardware interfaces for communicating between application program and hardware. It simply avoids the OS middleware altogether to operate directly on the hardware. The hardware interfaces provided in Bare Machine Computing applications are very similar to system calls and kernel interfaces. Thus, we claim that we should be able to transform any OS or kernel based application to a bare machine application. This hypothesis will be investigated during this research and the proof of it will be validated.

3.3 Approach

The approach taken to demonstrate the above hypothesis stems from the background and experience gained in building BMC applications over the past decade. In the big picture, there is a plan for four stages. (1) The first step involved selecting a large application (that is written in C/C++ code only), and transforming it to run on a bare PC; this process enabled us to understand the intricacies and hurdles involved in the transformation process. (2) In the second step, transformation tools were designed and built that helped this process. (3) In the third stage, the transformation process was studied and its net effect regarding gain in performance or loss of functionality and

reduced code sizes. (4) In the final phase, transformation strategies and methodologies are laid out for further research in this area.

Chapter 4 SQLITE DBMS STRUCTURE AND CHARACTERISTICS

SQLITE DBMS application is chosen to demonstrate the transformation process as it is a very large C programming application and consists of complex structures and styles. SQLITE is a standalone single user database engine which runs on Windows or Linux operating systems. It is a commonly used database system across many academic institutions. It has an amalgamation package which consists of two files; shell.c and sqlite3.c. The size of shell.c and sqlite3.c is 86, 016; 4,323,826 bytes respectively. The total number of lines of code in both source files is 129,003, commented lines of code are 55,691 and executable statements are 40,297. This code has numerous code complexities such as over 6200 lines of code inside the scope of a switch statement, hundreds of macros, and user defined OS related functions. There are dozens of pre-processor statements that include and exclude various part of the code. We used Windows version of the code that runs in Microsoft Visual Studio 2010 Express environment. When the application runs, a user can perform standard database functions such as 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 can be run on a bare PC without using any operating system.

Chapter 5 BARE MACHINE COMPUTING AND INTERFACES

The transformation process used in this dissertation was motivated by BMC paradigm [14] and its related applications. It is also very much at the background of the development of bare PC applications and in making direct hardware interfaces available to programmers. One can rightly think of BMC as a converging point for all bare PC applications, whether developed from scratch or created through the transformation process. This dissertation validated some of the BMC paradigm concepts and developed new interfaces that are relevant to the transformation subject, which is addressed in this research. It is essential, therefore, to understand BMC connection to the transformation process.

5.1 BMC Computing Paradigm

BMC paradigm was originally referred to as dispersed operating system computing (DOSC) [14], but further evolution of DOSC into BMC concepts has occurred over the years. Fig. 1 illustrates concepts of an application object (AO) [12] and describes the building blocks for a BMC application. A conventional OS, kernel or embedded software acts as middleware between the hardware and an application. An application programmer is isolated from an application's execution environment, resource control and management.

That is, the programmer has no direct control of the program's execution or the resources needed. In the BMC paradigm shown in Fig. 1, the OS is eliminated; the AO programmer is totally capable of directing application execution, control and management; the AO programmer has knowledge and full control over a given application as well as its execution. The BMC paradigm differs from conventional

computing in two major ways. First, the machine is bare, with no existing software and protected resources. Second, an AO programmer controls the program's execution and manages the hardware.

The BMC paradigm makes a computing device owner-less and simplifies the design of secure systems since there are less avenues of attack and no underlying middleware that an attacker can control. Viewed another way, when a device is bare and contains no valuable resources such as a local hard disk or kernel, there is nothing to own or protect. In BMC, mass storage is external and detachable. The mass storage can also be on a network. In this approach, an AO is built for a given set of applications to run at a time on a machine as a single monolithic executable. The boot, load, executable, data and files are stored on a mass storage device such as a USB. When a USB is plugged into a computer, the machine boots and runs its own program without using any extra software or external programs. This implies that no dynamic link libraries (DLLs) or virtual machine code are allowed in this approach. What runs in the machine, is exactly what has been loaded (and nothing else).

This computing paradigm is again different from conventional computing approaches since it is based on applications instead of computing environments. This is not a mini-OS or kernel, as there is no centralized program running in the machine to manage resources. Instead, the resources are managed by the applications themselves and run without using any OS/kernel or intermediary software.

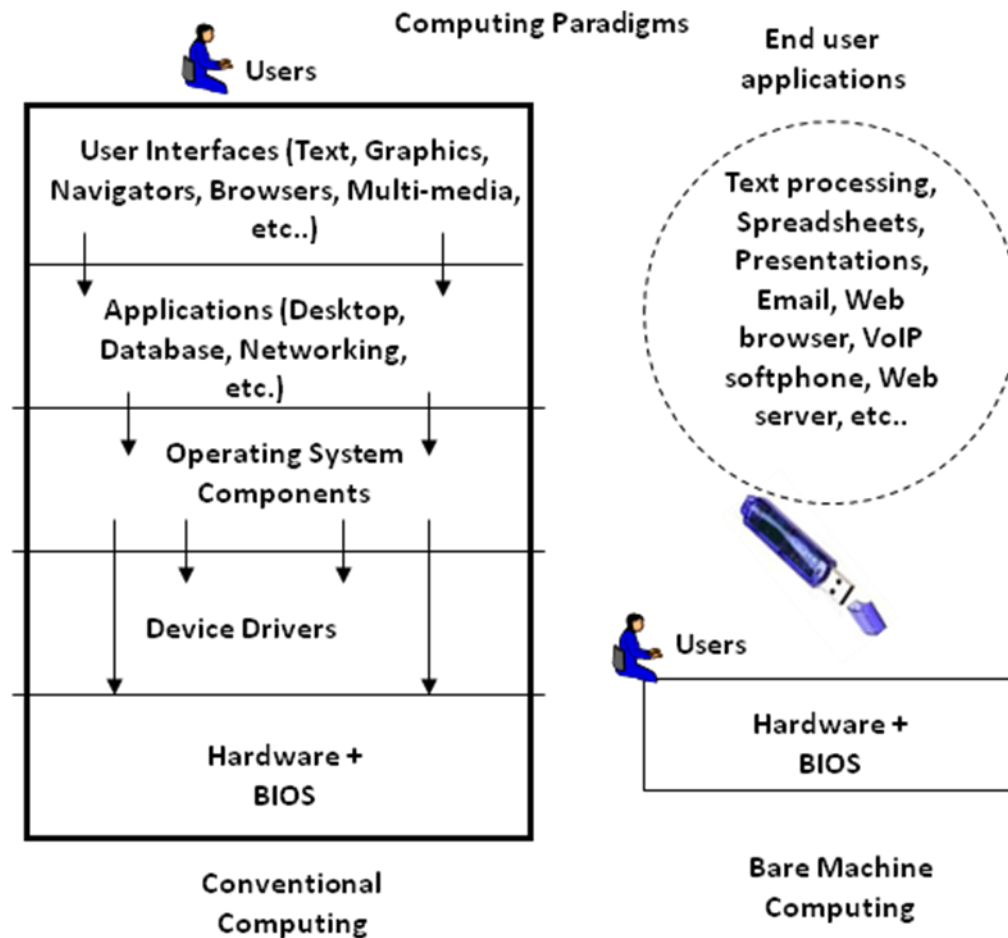


Figure 1 Conventional OS and Bare Machine Computing

5.2 BMC Application Development.

In BMC, a suite of applications such as a text processor, Webmail server and Web browser can be bundled together and run without any OS or kernel support. Fig. 2 illustrates the major steps involved in developing BMC applications. First, a choice has to be made about the suite of applications; next, the architecture of the CPU on which they will run has to be identified. Using today's CPUs, constructing a BMC application is a daunting task because they provide neither direct hardware interfaces nor support for a development environment that facilitates building applications independent of an OS. For

example, a bare PC requires the BIOS to boot and an ARM processor requires a Uboot tool. The program counter (PC) of a given processor is not directly accessible to the programmer. In an x86 architecture machine, a program counter can only be loaded by jumping to the task segment, where the PC value is stored and updated by the CPU. In a BMC application, the PC must be handled inside an application and not controlled by middleware.

Memory requirements must be considered for the code, data and stack of a given application. It is necessary for the application programmer to do this as these applications run in real memory, which is cheap and affordable today, therefore making it feasible to avoid paging, virtual memory overhead and management. The absence of any middleware or an OS in the system eliminates other OS features commonly found in today's technology. Most BMC application suites therefore require small amounts of memory compared to OS-based applications. For very large applications, one can use mass storage to provide extended storage using swapping techniques. Section 5.5 describes details of the memory map created for some real world applications using the BMC paradigm.

The next step is to construct the application suite using programs that are independent of any OS. The application programs should be able to run on any compatible CPU without changes or adaptations. Different CPU architectures have different compilers to compile code. This requires identifying I/O related code and deploying direct hardware interfaces. One of the key elements in writing BMC code is being able to differentiate between code that is OS dependent, code that is OS independent and code that is I/O related. For example, file I/O is OS dependent code and a for-loop is OS independent

code. User interfaces to support keyboard, mouse and display are all I/O related code. Once OS dependent code and I/O related code are written (as hardware interfaces), they can all be integrated with the rest of the OS independent code and run as a single monolithic executable. The above approach poses many challenges in developing BMC applications. They include the boot-up process and loading of an application suite. Each computing device is different in its boot process and the internal details are often hidden. Similarly, loading an application on a bare device also poses difficulty as it requires readily available tools that are OS dependent. Developing an OS independent loader requires thorough knowledge of the CPU architecture and its development environment..

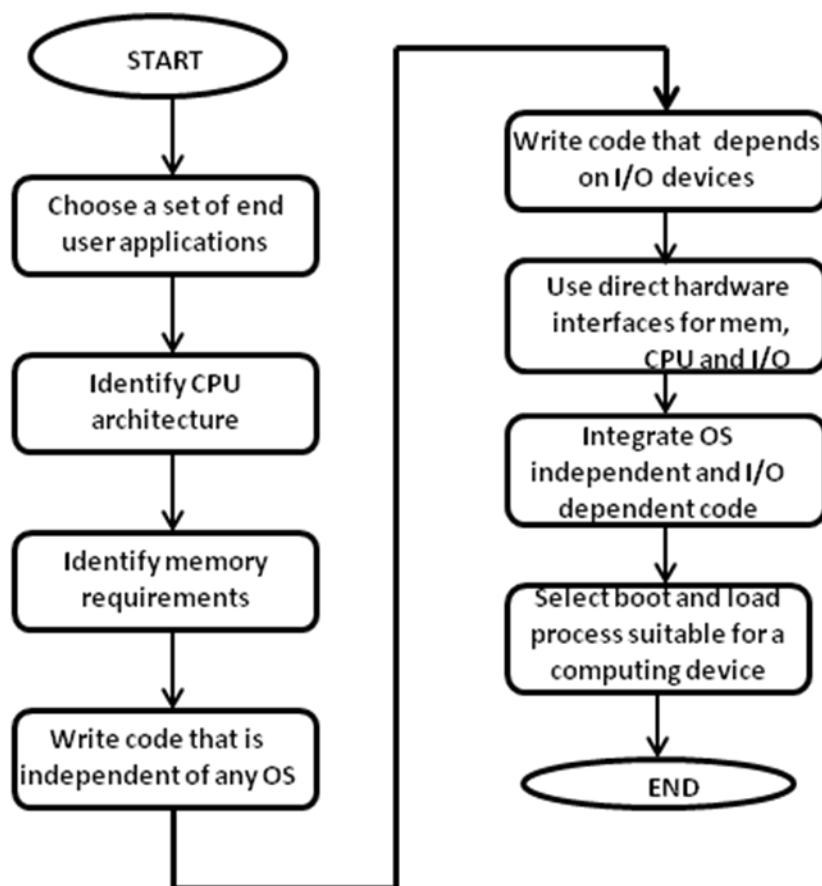


Figure 2 Steps in Developing Bare Machine Applications

Domain knowledge and some expertise for each CPU device are required to develop the bare boot and load processes.

5.3 Direct Hardware Interfaces.

Conventional computer applications and programming languages use OS calls or system calls injected at link time from an OS such as Windows [27] or Linux [9]. These calls include memory, keyboard, terminal screen, network, mass storage, and interrupts. There are also modern OS systems that include in their repertoire other commonly used OS independent functions such as memory copy, string operations and concurrency control as if these are system calls. Today's computer applications and the programmer expect these calls or interfaces to be included at compile and link time by a given compiler and linker. Direct hardware interfaces related to SQLITE DBMS were developed as part of this dissertation as described in [20].

5.3.1 Static and Dynamic Memory

Static memory needs depend on the size of code, data and stack needed to run a program. When an executable is created, this information is available to the programmer. Thus, for a given executable, one can specify its requirements for memory. An AO can also be designed so that it can read the existing memory and restructure its code, data and stack in real memory and external mass storage or network. The code image is small as there is only one AO running at a time in the machine, and applications that are related are grouped to run together.

Dynamic memory needs are however not known until run time. In a bare machine application, an AO programmer estimates the dynamic memory. Appropriate exceptions for memory can be set to manage dynamic memory; when large dynamic memory needs

arise, one can use secondary storage in place of large dynamic memory. System calls similar to `malloc()` and `free()` can be designed to support dynamic memory management. One can allow the memory controller to communicate with an AO and thus provide appropriate memory interfaces to manage memory in the AO. As the memory technology improves and becomes cheaper, it is also conceivable to assume full address space (4GB in a 32-bit architecture) in a machine to avoid all memory management issues and provide direct control to a given AO.

5.3.2 User Interfaces

The most common user interfaces are keyboard, mouse, touch-screen and terminal screen. These resources are managed by the OS in conventional systems. In bare machine applications, keyboard interfaces are part of an AO where the keyboard interrupt code places the data in a user buffer. Similarly, mouse data is also placed in a user buffer. An AO programmer designs the code to directly interface with a keyboard or a mouse. The terminal screen is usually controlled by a video memory or its graphics adaptor. An AO programmer can directly store output in video memory or write a bare video driver to control the screen. All device drivers supporting a bare application have to be bare and provide direct hardware interfaces to applications. They cannot be hidden from the application programmer as it is done in an OS environment. Other user interfaces have to be handled in a similar manner to the above interfaces.

5.3.3 Network Interfaces

Most ordinary computing devices today have one wired and one wireless network interface. The device drivers for a network interface are controlled by underlying OS. Bare machine device drivers that provide direct network interfaces to an AO are needed

in BMC. Instead of current OS dependent network drivers, an AO programmer can initialize a network driver, configure relevant internal registers, and read or write to buffers and control registers. Such a design allows direct communication to applications and avoids the need for any middleware. As the drivers are now encapsulated within an AO, the network hardware is not accessible to other applications when a given application suite is running in the machine. A bare PC USB device driver and its implementation is described in [13].

5.3.4 Process Interfaces

Many computer applications require process creation, deletion and management, which are usually controlled by an OS. In Intel x86 processors, process control and state are maintained by the CPU in a task segment. Interrupt gates are used to switch from one task to another. That cannot be done in a bare environment since these interfaces are accessible to an AO programmer. Control of the CPU is placed in an application program for creation of a new process (or a task). The global descriptor table (GDT) and local descriptor table (LDT) entries are used by the AO programmer to control task memory. Thus, when a machine becomes bare, the CPU and tasks are managed by an AO programmer. Task management in a bare machine is much simpler than in a conventional system, and the code size is also smaller compared to an OS managed system. Conventional Web server systems are complex and may create over 7000 tasks (in an x86 box) to provide high performance [10]. Process interfaces can be generalized in the near future and made available to an AO programmer for any given CPU architecture. Current systems hide all these interfaces under an OS or some form of similar middleware.

5.3.5 File Interfaces

In conventional systems, a file system is part of the operating system. File systems use some standard specifications such as FAT32 or NTFS. Files can be transported across multiple operating systems and applications if they use standard specifications in their design. In bare machine applications, persistent data is under the control of an AO programmer and the data itself is part of an AO. Programmers can use their own file storage specification or use a standard specification to transport files to non-bare systems. One can also do a raw file system in an AO to avoid all file management complexities and hide the files within an AO (the only visible AO to the files). This may be the most secure way to implement a file system. File transfers can also be accomplished through a network or by message passing. A given file system interface uses a bare device driver and controls the relevant device operations.

5.3.6 Boot and Load Interfaces

Boot and load facilities are usually under the control of the OS and the underlying BIOS calls. In BMC, these interfaces are controlled by the AO programmer to facilitate bare machine applications. Soft and hard boot can be used to control the machine when needed in bare machine applications. These interfaces also vary across platforms; ideally, a standard boot and load mechanism to run bare machine applications across multiple CPU architectures and machines is the best solution (what is described in Section 5.5 is a method that has been implemented for x86 Intel CPUs)

5.3.7 Compile, Link and Library Issues

Compilers and linkers generate different formats for executable, which pose problems in loading and running bare machine applications. There is a need for homogenization in

these tools to develop common bare machine applications that can run on many pervasive devices. New programming tools can be developed to compile bare machine applications using existing libraries and batch files, or new features can be added into existing Microsoft Visual Studio and Eclipse development tools to provide bare machine compilation options. Common libraries such as string operations, memory operations, locking, shared memory, message passing, and concurrency control are OS dependent and part of the OS libraries. However, they can be generalized and designed to run across many CPU architectures.

5.4 Bootable USB

In the BMC paradigm, applications are carried on a removable storage medium such as a CD/DVD or a flash drive. This device also carries a boot program to boot and load its own application object suite. A typical way to create a bootable USB is as follows. A bootable USB is created using a special tool written in C and assembly language. This tool is a batch file that runs in a DOS window. The USB is formatted for FAT32 before its use. The bootable USB should have three files as shown in Fig. 3(b). The boot file is stored in the boot sector (**#0**), the prcycle.exe file is stored at **0x3be000**, and the application file (shown in Fig 3(a) as shell.exe), is stored at **0x3c4000**.

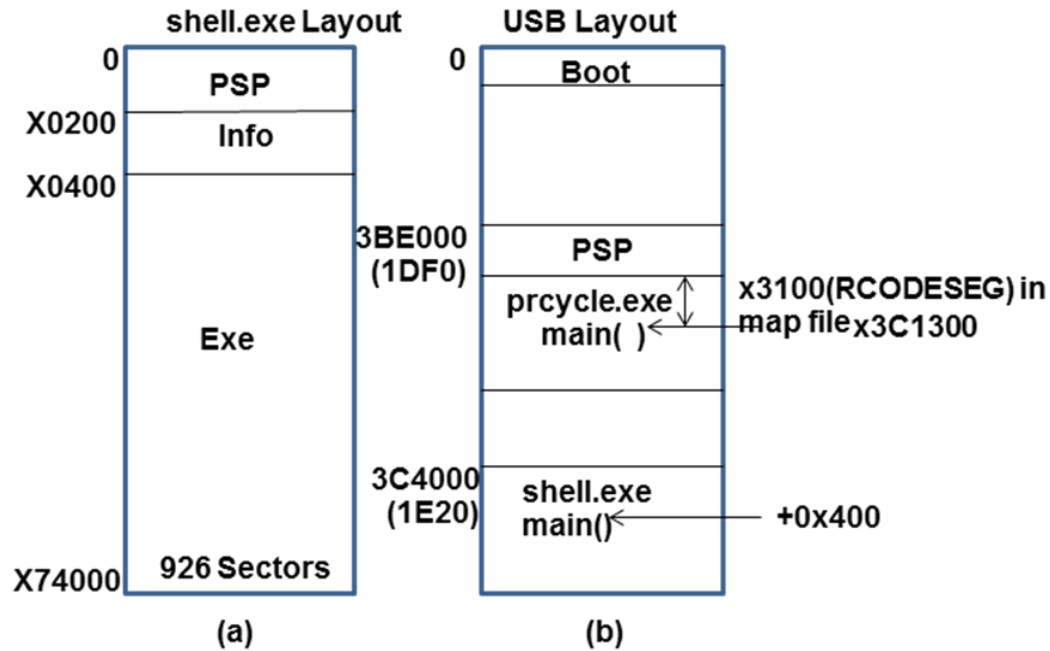


Figure 3 USB Layout

The prcycle.exe file (22, 037 bytes in size) contains assembly code to boot a bare PC, provides the user interface/menu, and facilitates the loading of AOs (in this instance, shell.exe). It enables the switching from real to protected mode and vice versa for handling low-level interfaces. It also contains, IDT, GDT, TSS and BIOS interrupts to provide the AO programmer with direct control of the CPU. This part of the application code thus plays a key role in enabling the programmer to manage the hardware resources in a bare PC. In summary, the batch file copies files onto the USB, installs a boot program, and creates a bootable USB. This entire process does not require any software other than what resides on the USB (and is thus part of the bare PC application). There is no dependence on any specialized commercial tool or software. This enables BMC applications to be independent of any OS-related environments and tools. It is also possible to use existing boot tools to create a bootable USB; however those tools must guarantee high security if needed in a system. The approach proposed here demonstrates

building bare machine computer applications in a single environment where every aspect of software development is controlled by an AO programmer with no other dependencies. This approach facilitates enhanced security to computer applications.

5.5 Memory Map.

As discussed in section 5.2, the AO programmer needs to design the real memory layout when developing a BMC application. Fig. 4 shows a typical memory layout for a given application suite. An AO programmer prepares this map before designing a given application suite. The prcycle.exe program is used on the bare platform to load the AO at **0x600** in real mode memory. The main() entry point for prcycle.exe is located at **0x3100**, which can be obtained from the prcycle.map. When the PC is booted, it must jump to **0x3900** as instructed by this memory map. A user loads the example application (shell.exe) by using the menu provided by prcycle.exe (not shown here). The executable for this AO is loaded at **0x0011E00** as shown in Fig. 4. The reason for using this particular address for loading shell.exe is discussed below. Visual Studio 8.0 (and later editions) of compilers behave differently than the previous versions when generating an exe file. In previous versions, when the entry point in shell.map indicates **0001:00000000**, it usually implies that the main entry point in shell.exe is at **0x1000**. In newer versions, this is not the case. In Visual Studio 8.0 (C++ versions), the executable starts at address **0x400** instead of at **0x1000**. In Fig. 4, the AO (shell.exe) is located at **0x0011E00**. The higher 16-bit address **0x0011** indicates that it is loaded above **1MB** to load it in a protected mode memory address.

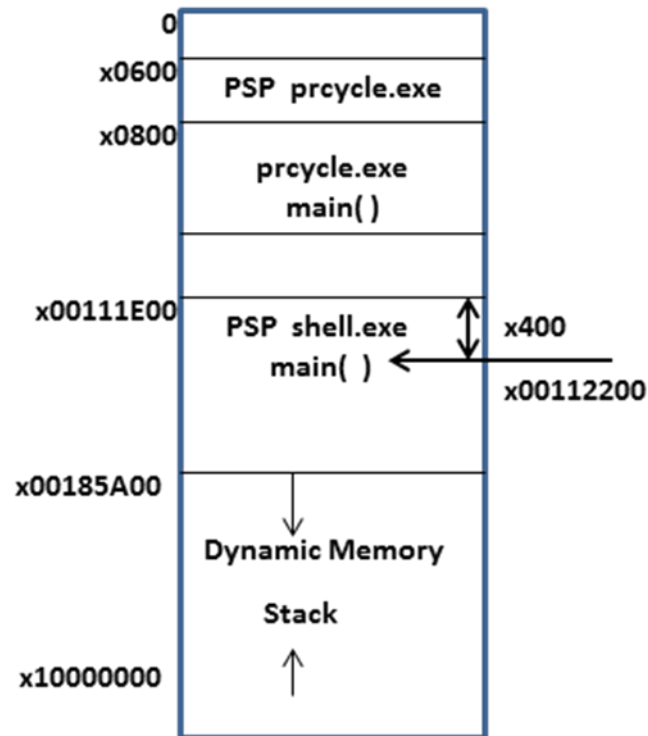


Figure 4 Memory Map

The lower 16-bit address **0x1E00** is derived as follows. The compiler start address for shell.exe is **0x0000**, but it actually starts at **0x400**. It was observed in the executable that the offset used by this compiler is **0x1e00** more than the actual offset in the executable. Thus, when the executable is relocated at **0x1e00**, the references to the variables were correct as it was generated by the compiler. The main entry point for shell.exe should be at **0x1e00 + 0x400** – see Fig. 4. A generic tool is needed to resolve such intricacies involved in generating a memory map for a mass storage device. This tool should consider compiler options, executable formats and map files to create a memory map that is suitable for a given bare machine device.

To summarize, this section described, in general, the development of BMC applications and identified the generic direct hardware interfaces needed to develop these applications. It also illustrated a bare machine application architecture that enables a

BMC device to be used for many pervasive applications. This approach will save time, energy, resources and the cost of developing applications for each pervasive device. The BMC approach enables these hardware interfaces to be incorporated in the hardware thus making the latter more intelligent and able to communicate with the software. The proposed hardware interfaces were used to construct complex BMC applications proven to be small in code size, simple to use, yielding high performance, and inherently secure in design. The BMC paradigm demonstrates a new approach to future computing based on completely self-supporting applications that eliminate all forms of middleware. This paradigm also served as a backbone to conduct code transformations as described in this research.

Chapter 6 TRANSFORMATION STRATEGIES

Software is routinely ported from one platform to another as cited in [6]. What is the difficulty then in porting SQLITE DBMS to run on a bare PC system? When an application is ported from one OS or kernel-based platform to another, we need to only worry about the OS discrepancies in implementation as the basic structures are similar across many platforms. When porting a given application to a bare PC, many challenging issues arise depending upon how the original application code was written. We generally find that if the code has minimal OS-dependencies, it is significantly easier to port it to a bare PC or a bare machine in general. We also find that it is not very difficult to write code that is OS independent; however, most application programmers do not have or see a need to do this. If a conventional application is developed that needs to be run in future on a bare system, it is beneficial to write the original code with little or no OS dependencies. Then most of this code is easily ported across many pervasive computing devices with only minor additions or modifications. For example, we have ported the code for numerous security algorithms from Windows or Linux to a bare PC with minimal effort [7] because such security code is much less dependent on the OS than would be the case for other applications. We have been hitherto unsuccessful in our attempts to directly port drivers, Web servers and e-mail/Webmail servers that run on OS-based (e.g., Linux-based) systems. Consequently, we have had to design and implement our own bare PC code for the above applications.

The code transformation process described here should not be confused with the notions of ubiquity and/or portability as in the ability to run a Java application anywhere using the Java run-time environment (JRE). A transformed bare PC application needs to

run directly on the bare hardware without any operating system/kernel support or any form of centralized control. In contrast, Java programs require a Java Virtual Machine (JVM) and byte code interpreters and loaders. If a Java program is written without any OS dependencies, and a bare JVM is available, then one can readily port Java applications to run on a bare PC. However, building a bare JVM is itself a formidable task and we decided not to pursue transformation approaches that deal with Java.

We also investigated using a virtual machine on Linux and trapping OS-related entry points and environment variables. We did not continue this approach since modern Linux systems are typically quite complex and the effort for us to build even a small kernel with the necessary properties would be considerable.

As can be seen from the above discussion, there are many similar strategies that can be potentially used for the bare transformation. We evaluated many of these, deliberating their respective pros and cons. We also developed an informal system of “passing or failing” a given strategy depending on the results of our evaluation. At the end, we were left with just the following four strategies that are shown in Fig. 5.

6.1 Same Executable

One can use the same executable as the one built in Microsoft Visual Studio, which runs on Windows. This executable includes or carries all the system libraries and its computing environment (environment variables). This is not directly loadable or executable in a bare PC as there is no OS or kernel running in the machine.

In a bare PC system, we can use a bootable USB (with our own boot code on it) that can be used to boot the PC. Once the bare PC boots, we can load the E1 executable into memory through a simple menu interface. From the menu, we can also jump to the main

program in the shell.c file. This methodology has been and is still used with success in numerous bare PC applications. However, those applications were originally written following the bare PC programming paradigm.

The SQLITE DBMS executable contains numerous system calls, Microsoft Windows attributes, data types, and timers/interrupts that cause the executable to hang immediately since there are no system libraries for resolving them. Even though, we made some progress using this approach, we ran into many debugging issues. It was very tedious to dive into the internals of SQLITE DBMS to debug the code. We also tried to do reverse engineering: using the same code running on Visual Studio and inserting break points in the code. This approach immediately failed as the code uses standard OS-based mechanisms and techniques including caching, paging, dynamic memory, locking, mutual exclusion logic, access mechanisms, threads, file system, and memory mapped files. This approach requires a thorough understanding of the Windows-based code to debug and make it run on a bare PC. Considering these issues, we abandoned the use of this approach for transformation.

6.2 Trap System Calls

Using the same executable, one can load it into memory using the bare PC menu and loader as described above. Whenever a system call causes an interrupt, it can be trapped in the AO and appropriate bare PC interfaces can be substituted. This approach assumes that the transformation only requires the substitution of bare PC interfaces for system calls. This assumption is not valid because SQLITE DBMS contains a variety of compile options, pre-processor statements, and optimization parameters, and requires local cache management as well as local **memory** allocation/management. In essence, the SQLITE

DBMS application creates its own “**mini-OS**” within the application to achieve higher performance and to control resources optimally. Thus, trapping system calls alone is not sufficient to achieve a complete transformation. Moreover, trapping system calls in a bare PC is by itself a daunting problem since one has to be aware of all the (numerous) Windows system calls. The SQLITE application, like many complex applications, incorporates its own system program features, and can do its own memory, cache and thread management. We found that it is very difficult to isolate these system dependencies, resulting in a situation with issues similar to those discussed in 6.1 above.

6.3 Resolve at Assembly Level

It is also possible to attempt code transformation at the assembly level. Assembly language translators can be used to build tools to translate from one processor type to another. For example, one can use DisIRer tool [11] to translate from x86 to ARM. However, such tools do not consider OS dependencies that have to be eliminated to enable the code to run on a bare PC. Furthermore, assemblers provide little help at any level higher than the machine language level. This approach requires a thorough understanding of Microsoft MASM assembly language and knowledge of the internals of compiler implementation: passing parameters, register layout, interrupts, base and limit registers, real and protected modes, TSS (task segment state), and kernel and user modes. It also requires knowledge to control the PC at the lowest practical level (the operating system, the BIOS, and even the hardware level where necessary). We did not consider this approach further due to such issues.

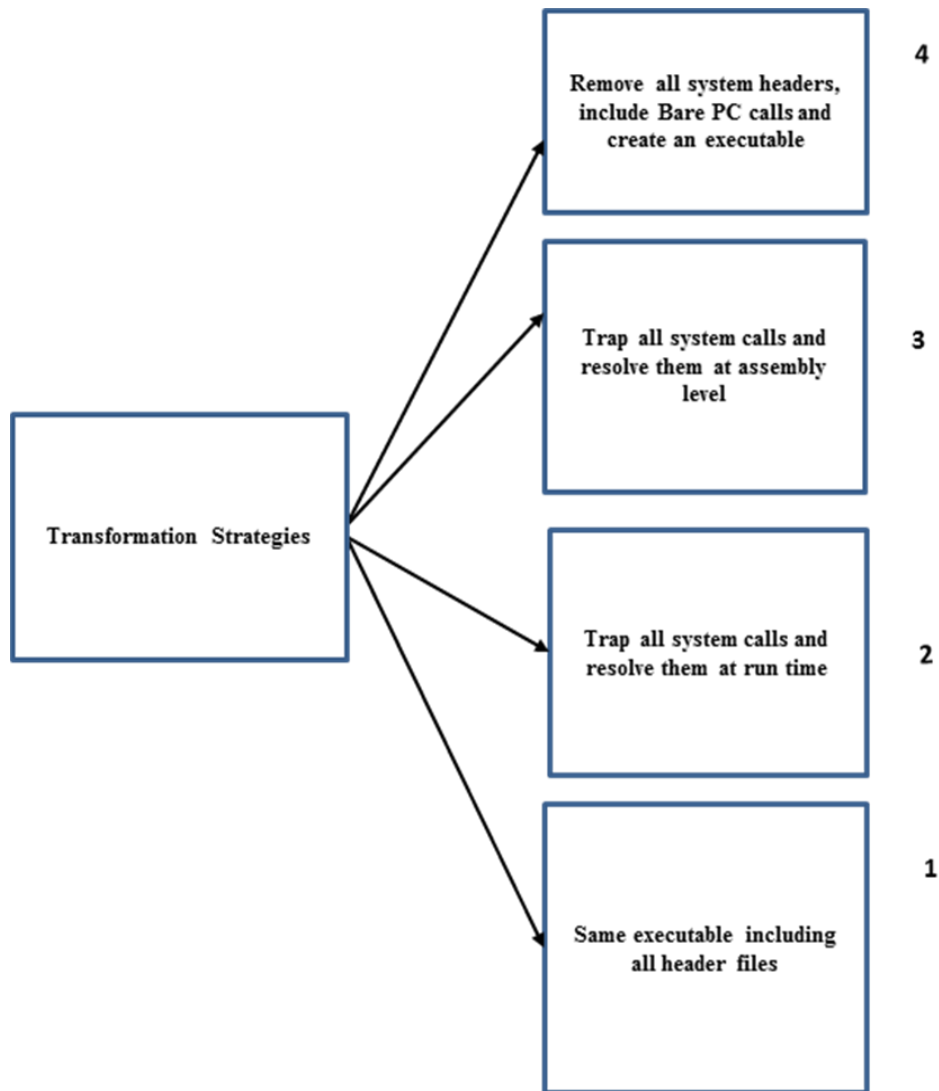


Figure 5 Transformation Strategies

6.4 Remove all Header Files

BMC applications run on a bare PC without any OS or kernel support, and thus do not have any OS-related interfaces. This implies that one should build the application without any OS-dependent header files in it. Once all such header files are removed, the resulting program has to be compiled and linked. SQLITE uses many kernel data structures and OS-dependent constructs: variable arguments, mutual exclusion, threads, and others

noted earlier. Some header files are OS dependent and some are not. This approach requires that we identify data types and functions that are OS dependent, and replace them with bare PC types and structures.

After considering the above strategies we had chosen the last option to transform SQLITE DBMS by eliminating header files.

Chapter 7 TRANSFORMATION PROCESS

The transformation process involves making the SQLITE DBMS code to compile and run on a bare PC. We have studied many strategies as initially proposed in [19]; these strategies have been reviewed again in section 6, but they did not result in a successful transformation. Many challenges are encountered when transforming large OS-based applications with complex code to run on a bare PC (testing, validation and debugging can pose problems especially since there is no environment such as an IDE to support bare application development, and only a few primitive tools that can run on a bare system). Some issues are partially resolved by using the Visual Studio (VS) environment for testing, validating and debugging the code during the transformation process. For example, we coded methods and bare PC interfaces in a fashion that is independent of the OS and tested such pieces in the VS environment using the Additional Library override feature. The VS environment does not allow the complete bare machine code to run, as it requires resources from its Windows environment through system calls. It is difficult to eliminate all system calls when using the VS/Windows platform. For example, memory allocation (`malloc()`) uses virtual memory and it is obtained from heap space. In a bare PC, this is physical memory; it is allocated and controlled via the AO code by the bare software developer (the file system is also managed by the application if it is required). There are hundreds of header files included in a Windows program, even if the application program does not require all of them. The header file “Windows.h” is an example for this.

The general transformation process model is shown in Fig. 6. VS 10 (C/C++ compiler) is used as the development platform for the bare PC application, with batch files to compile bare PC programs. The bare PC hardware API to support the application is also built during the transformation process (some interfaces may be reused from other bare applications). The main objective behind the transformation approach is to eliminate OS dependencies without understanding code details relevant to application logic, and to minimize changes to the original code.

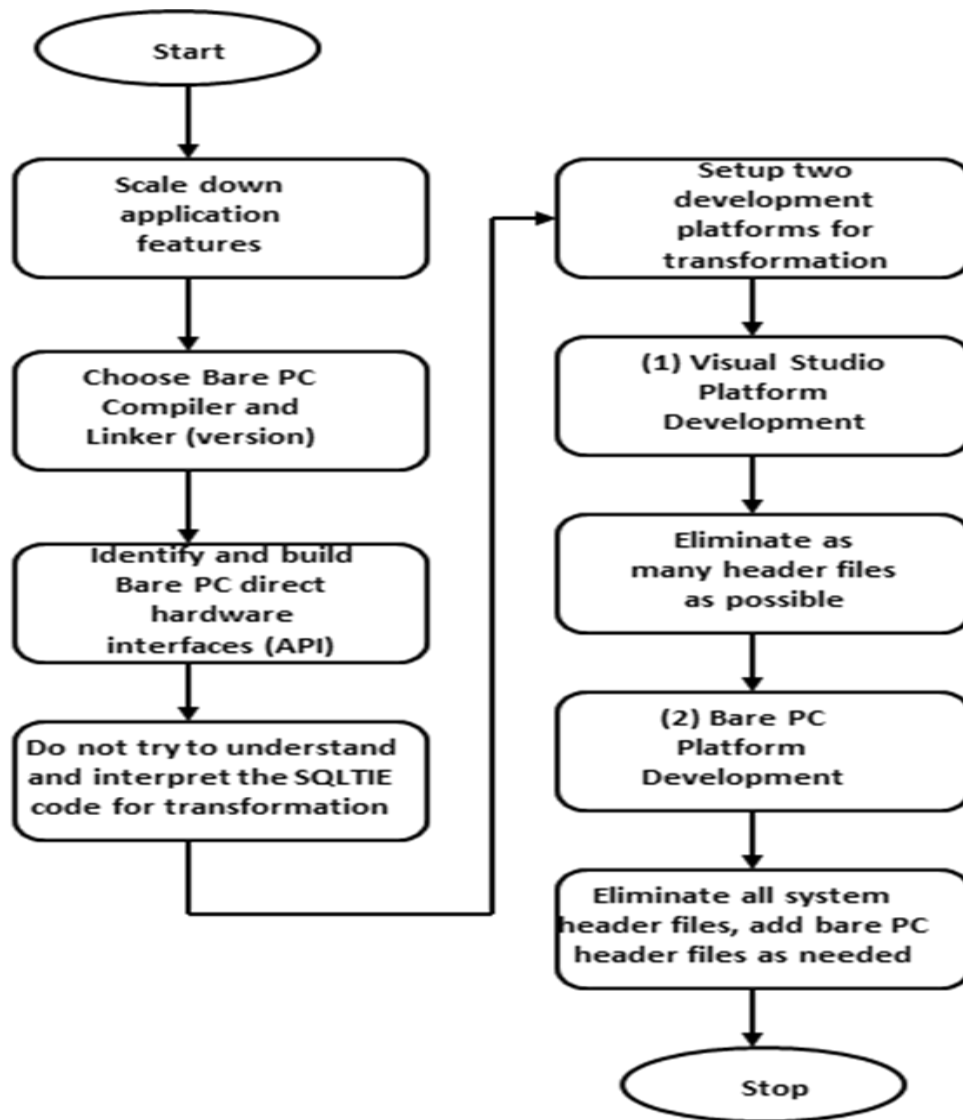


Figure 6 Transformation Methodology

Two development environments were set up: one for the VS application and the other for the bare PC application. In the VS application, as many header files as possible are eliminated; this code is then used to transform to a bare PC. Then the remaining system header files are removed by adding the bare PC interfaces and fixing any bare PC-related issues until the application runs successfully on a bare PC. This means the bare PC application now runs and has the same results as the OS application. More details regarding the transformation process are given below.

7.1 Scaling Down Features

By scaling down some functionality in SQLITE, the transformation process is simplified. For example, an “**in-memory**” database was used to eliminate file-related code in the transformation. Floating point and shared cache options were also turned off. Complex concurrency and locking mechanisms were not used since these are avoided in the BMC paradigm and illustrated in bare PC applications [3, 10, 21]. The scaled-down options apply to both the VS and the bare PC SQLITE applications during transformation.

7.2 Visual Studio Application

For the VS application, baseline code was downloaded from the SQLITE Web site and scaled down as noted above. A test case suite was developed to test query results that includes standard commands such as create table, insert (multiple) rows, and select table. This suite was used to test correctness of database operations after each step of the transformation in VS. Fig. 7 shows the steps in this transformation process, whose goal is to remove as many OS dependencies as possible.

VS offer numerous compile and link options that help to transform an application so that it runs on a bare PC. For example, the **NODEFAULTLIB** option is used to identify system calls that are in the application. All these system calls must be resolved in the bare PC application to make it run without OS support. There were 85 unique system or library calls in SQLITE DBMS. These system calls/library interfaces can also be obtained from a linker when **NODEFAULTLIB** option is used, which show up as link errors. These link errors are due to different types of system calls: we categorize them as shown in Fig. 8. Some of these calls will show up as duplicates in the original link error list as they are referenced in two different source files (shell.c and sqlite3.c). 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) [9]. 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 [15] which are controlled by an AO programmer.

There were additional dependencies due to Link options, which were handled by including a bare PC user library to be used during compilation i.e., bare PC direct hardware interfaces were put in a library (**rkkvs.lib**) to replace the original system libraries. The Assemble Machine Code and Source Listing option, **/FAcs**, is used to generate the assembly listing and .asm files (this is very useful to understand the role of system calls in the code).

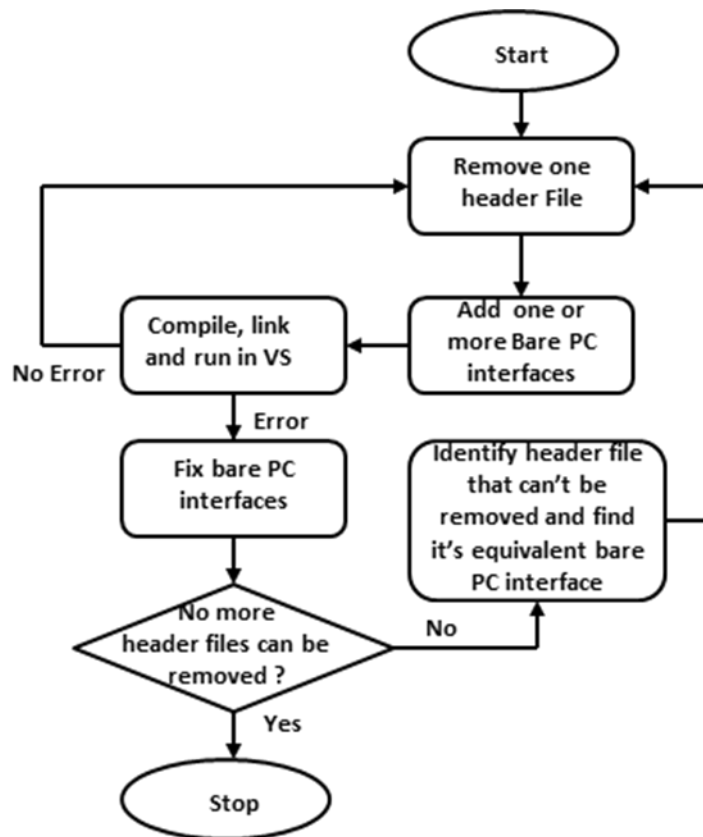


Figure 7 Visual Studio (VS) application platform

The process to transform the VS application is similar to that of developing an ordinary C/C++ application (except that the main focus of transformation is to eliminate all system calls/libraries). One header file at a time in the VS application is removed by commenting it in the source code. When the program is compiled and linked, it shows the missing system calls in the application. These calls are replaced with bare PC direct hardware interfaces and recompiled. Only one system call at a time is resolved since the bare PC interfaces are not yet fully tested. When the bare hardware API becomes more robust, it will be possible to resolve multiple (or all) system calls together to speed up the transformation process. The VS application uses a large number of libraries and DLLs. It

is necessary to guarantee that the system calls handled by the bare hardware API are used by the linker.

These system calls or interfaces used come in three forms: **(1)** the name of the call has single underscore and it is explicit in the code (e.g `strcmp()`, `_strcmp()`): in this case, all `strcmp()` methods in the source code must be replaced by `AOAstrcmp()` to guarantee the usage of the bare API; **(2)** the name of the system call has a double underscore and it is not explicit in the code (e.g. `__allmul`): in this case, this call must be added to the bare PC library and it must be removed from the system libraries; **(3)** the name of the call sometimes is simply a constant such as `__fltused`: in this case, that constant and its value are provided if needed. During the transformation, system header files are removed and bare PC hardware API are added until no more header files can be removed. Some header files invoke other header files and invocations may be indirectly recursive. For example, it is not possible to remove the “windows.h” file during VS compilation.



Figure 8 OS related Calls and Library Functions

In general, every system call needs a header file, but every header file may not have a system call. There can be different types of header files: e.g., user header files, constant header files, and structure header files. The 85 system calls in the SQLITE application, as shown in Fig. 8, were classified into the following types: 8 Arithmetic Assembly, 2 Disk Management, 2 Standard I/O, 1 Error, 27 File Function, 2 Floating Point, 1 Object Handle, 3 Library, 10 Memory, 3 Process, 1 Stack, 4 String, 2 System, 10 Timer, 7 Type,

and 2 Unicode and Character Set calls. According to [9] and [27] respectively, Linux and Windows systems have three to four hundred system calls.

The header files that could not be removed were: `windows.h`, `stdarg.h`, `stdio.h`, `stdlib.h`, and `assert.h`. Since we scaled down the application, we were able to provide all bare PC hardware interfaces except for `malloc()`. In VS, `malloc()` call gets memory from memory management and it is a virtual memory with paging. In a bare PC, all memory is real and the AO programmer manages it at program time. There were a large number of `malloc()` calls used in the code. We used the bare PC memory object to replace all the calls except for a single system `malloc()` to obtain memory as required by the application. The total memory obtained by this call was used by the bare PC memory object to provide memory for SQLITE DBMS calls invoked within the program. The pseudo bare PC code for SQLITE was then tested to verify that its results were the same as for the original VS code. Next, the transformed VS code was transferred to the bare PC to determine other modifications that were needed.

7.3 Bare PC Application

The transformation process on a bare PC is shown in Fig. 9. During the pseudo transformation in VS, system calls such as `malloc()` transferred from the VS system were eliminated and replaced with calls in the bare PC API. There were also other header files that could not be removed in the VS application.

Again, one header file at a time was removed from the list (`windows.h`, `stdarg.h`, `stdio.h`, `stdlib.h`, and `assert.h`) and replaced with appropriate bare PC interfaces until all header files were removed. The transformed SQLITE application was compiled, run and tested after each modification of the code to remove the remaining header files. In some

cases, header file removal required a new header file in the bare PC application that defines constants, variables, structures and data. In the SQLite transformation, we defined header files such as `sqlite.h`, `sqlite21.h`, `sqlite22.h` and `stdarg.h` for this purpose. In this case, once header files were removed and the VS system `malloc()` (just one system call) was used to provide memory, the application successfully ran in a bare PC. To verify success, we tested the bare PC application with a variety of create, insert and select statements and validated their functionality. We also checked that the results matched with the original VS application model for every instance of testing.

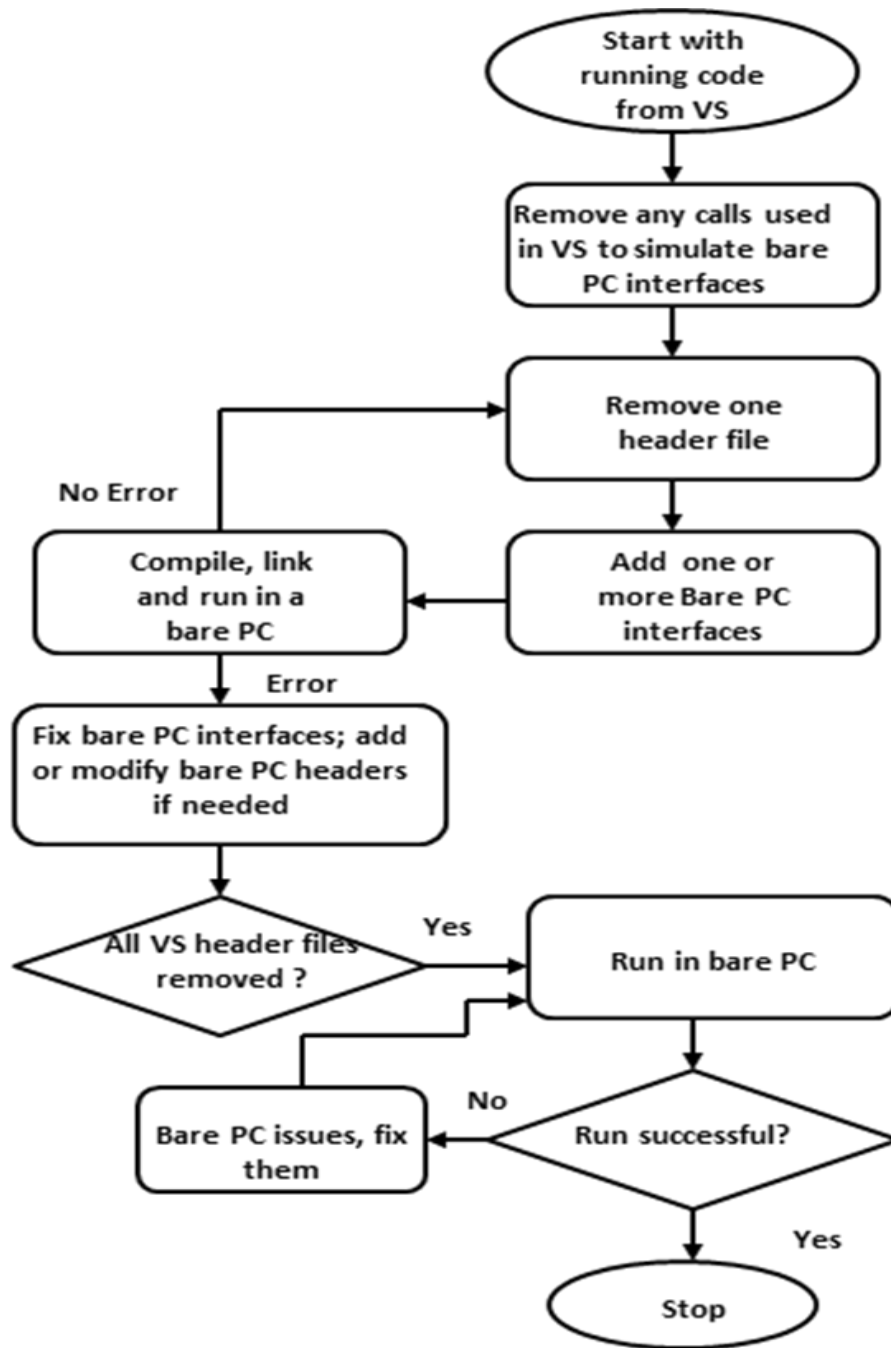


Figure 9 Bare PC Application platform

In general, once all header files are removed and replaced with appropriate interfaces, it should be possible to successfully run the application in a bare PC. If not, then there must be some issues with the bare PC interfaces or the transformation itself. Usually,

these arise from system problems related to the bare PC executable module and its memory layout. They may also be related to the bare PC loader and the BIOS, or the processing of system interrupts

```

..\bin\ml /c /Cx /Fl asmfiles.asm asmfilesb.asm
chkstk.asm

..\bin\cl /c /FA /ZI /Os /Gy /GS- shell.c sqlite3.c
cfiles.c aaa.c aaab.c memobj.c

...\bin\link /MAP /BASE:0x00100000
/NODEFAULTLIB /OPT:NOREF /MERGE:.rdata=.data
/OPT:NOICF /STACK:32000000 /LIBPATH:"rk\vs.lib"
/ENTRY:main shell.obj sqlite3.obj cfiles.obj aaa.obj
aaab.obj asmfiles.obj asmfilesb.obj memobj.obj
chkstk.obj

format e: /FS:FAT32 /q

copy prcycle.exe e:
copy shell.exe e:
copy data.txt e:

rwhd -m 5 newbootf32.bin

```

Figure 10 Batch Files

The **/bin** files from Visual Studio 10 Express Version are used for compilation and linking of the transformed bare PC application. The actual make file used to compile, link and generate a bootable USB for running the bare PC SQLITE DBMS application is shown in Fig. 10. Most of the statements in the figure are self-explanatory, except for the last statement. The **rwhd.exe** module installs a bare PC boot record after all other files are copied onto the USB. The USB is also formatted before copying the files. The

prcycle.exe file is the startup menu for bare PC applications, and the **shell.exe** file is the actual bare PC application (main component of the AO) for SQLite.

The **data.txt** file is a data file that can be used in the application for receiving additional parameters from the user. One can also store the persistent database files on the USB after they are used. The persistent database files are not shown here as we only demonstrated the SQLITE application with an “**in-memory**” database. A full-scale version of the SQLITE transformation would include persistent storage containing the database schema and data.

To summarize, the transformation process as described above used two steps: one using VS, and the other using batch files to compile bare PC code. It was **not necessary** to modify the SQLite application code, or understand the underlying application logic or the internal details of database structures. The transformed bare PC application was tested and validated for correct operation by running sample queries in a VS (Windows) environment, and on a bare PC, and comparing the results as shown in section 8. The methodology may be used to transform other complex applications to bare PC applications, making the code independent of any OS or environments. The currently manual transformation process could be modified in the future to build a tool for automatically transforming C/C++ or other programming applications. The methodology will also serve as a basis to transform applications that can run on a variety of systems and devices. The initial successful transformation of a complex application such as SQLITE DBMS indicates that future research into bare PC application development for performance, security, or other reasons may benefit from the new transformation methodology.

Chapter 8 RESULTS AND DISCUSSION

The SQLITE DBMS application is transformed to run on a bare PC (“**in-memory**” database) [21] with the methodology proposed in Figure 6. We did not modify any code, or understand any internal details of database structures or components in the code. After the transformation, the database application ran on a bare PC without any problems, as had been expected. The Visual Studio IDE was instrumental in completing the pseudo transformation phase, from which was output resulting source code that was very close to the bare PC source code. Fig. 11 and Fig. 12 show the same simple queries consisting of CREATE, INSERT and SELECT statements, executed on Bare PC and Visual Studio platforms respectively. Figure 11 demonstrates the creation of a t100 table with 6 columns and different data types on line 02. The parallel on the Visual Studio platform is shown in Fig 12.

In Figure 13, Line 03 shows an insert statement, with an error as it does not have “**values**” keyword in the statement. This demonstrates the parser detected a syntax error in the bare PC code. We replicate the same error on Visual Studio platform in Figure 14.

Lines 04-07 of Figure 15 show inserting of more values into the table successfully. Line 08 shows the creation of a different table successfully. Line 09 shows insert data into t200 table with syntax error (error message overwritten by subsequent query results) and Line 10 shows the corrected insert into the table. Line 11 shows more inserts into t100 table. Line 12 shows the select statement for t100 table. Lines 15-20 correctly print the query results for t100 table. Lines 23-24 show some debugging and testing display information. Results from the bare PC screen shot in Figure 15 have been complimented with similar results from Visual Studio in Figure 16. As noted in any of the screen shots

from the bare PC platform, the screen is divided into 8 columns and 24 rows to display the SQLITE outputs. A bare PC screen is, currently, a text-based window in bare PC similar to “**stdout**”.

Fig. 17 shows a complex query on the bare PC platform consisting of the creation of two tables and insertion of some data. The complex query on Line 10 shows a join and the printing of some attributes. The results of this query are correctly shown on Lines 15–18. The **.tables** meta-command in SQLITE is also tested and its results are shown on Line 22 (two table names: dept and employee). The sample screen outputs shown in Figs 11, 13, 15 and 17 demonstrate the correctness of the functionalities in the transformed code that runs on a bare PC; that is to say, functionality in the original source code has not been compromised in the course of the transformation. We also tested more queries and meta-commands to further validate the correct functionality of the transformed code.

These results serve to verify that this transformation approach - using the VS IDE to test, validate and debug bare PC code - is a viable approach. Since the model and methodology are very general, it is expected that they can be used to transform other complex applications. The scaled-down approach used in this paper will be extended to transform the full SQLite application with a file system and other features. The bare PC hardware API also needs to be enhanced to deal with other components of the standalone database engine and multi-threaded applications. The transformation process and the feasibility of it, demonstrate that it is possible to make existing applications to run on bare machines thus achieving a different form of ubiquity without using virtual machines. This observation infers a great potential for existing applications to make them

independent of OS or environments – especially when an automated tool is made available in the future.

```

01 1          2          3          4          5          6          7          8          00000003
02 create table t100(c1 int, c2 int, c3 int, c4 int, c5 int, c6 int);
03 insert into t100 values (100, 200, 300, 400, 500, 600);
04 select * from t100;
05
06 sqlite> _

08 c1          c2          c3          c4          c5          c6
09 100         200         300         400         500         600
10
11
12
13
14
15
16
17
18
19
20
21          1
22          C2 D2 E2                                1400D127 00000000
23          00000000 00000000 00000064
24 000000A8

```

Figure 11 Simple Queries QQ1 on bare PC

```

A6 : USERPROFILE/.sqliterc 0
SQLite version 3.7.6.2
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> create table t100(c1 int, c2 int, c3 int, c4 int, c5 int, c6 int);
sqlite> insert into t100 values (100, 200, 300, 400, 500, 600);
sqlite> select * from t100;
100|200|300|400|500|600
sqlite>

```

Figure 12 Matching results on Visual Studio for QQ1

```

01 1          SQLITE Trasformation, Towson University          00000002
02 2          3          4          5          6          7          8
03 create table t100(c1 int, c2 int, c3 char(20), c4 char(10), c5 int);
04 insert into t100(1111, 2222, 'Karne', 'Okafor', 1000);
05
06
07
08
09
10
11
12
13 sqlite> _
14
15
16
17
18
19
20
21
22
23 %s %s 00000001 near "1111": syntax error
24 00000000 C2 D2 E2 00000001 1400D12B 00000000 T

```

Figure 13 Error E1 flagged by Parser on bare PC

```

A6 : USERPROFILE/.sqliterc 0
SQLite version 3.7.6.2
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> create table t100(c1 int, c2 int, c3 char(20), c4 char(10), c5 int);
sqlite> insert into t100(1111, 2222, 'Karne', 'Okafor', 1000);
Error: near "1111": syntax error
sqlite> insert into t100 values(1111, 2222, 'Karne', 'Okafor', 1000);
sqlite> _

```

Figure 14 same error E1 on Visual Studio - see Fig 13

```

SQLITE Trasformation, Towson University
01 1      2      3      4      5      6      7      8      0000000B
02 create table t100(c1 int, c2 int, c3 char(20), c4 char(10), c5 int);
03 insert into t100(1111, 2222, 'Karne', 'Okafor', 1000);
04 insert into t100 values(1111, 2222, 'Karne', 'Okafor', 1000);
05 insert into t100 values(2222, 3333, 'Wang', 'Peter', 2000);
06 insert into t100 values(3333, 4444, 'Mike', 'Wijesin', 3000);
07 insert into t100 values(4444, 5555, 'Borg', 'Taylor', 4000);
08 create table t200(c200 int, c201 char(30));
09 insert into t200(8888, 'Table2');
10 insert into t200 values(8888, 'Table2');
11 insert into t100 values(5555, 6666, 'Eyer', 'Matyas', 5000);
12 select * from t100;
13 sqlite> _

15 c1      c2      c3      c4      c5
16 1111    2222    Karne  Okafor  1000
17 2222    3333    Wang   Peter   2000
18 3333    4444    Mike   Wijesin 3000
19 4444    5555    Borg   Taylor  4000
20 5555    6666    Eyer   Matyas  5000
21
22
23          00000000  1
24 00000000          C2 D2 E2          00000000 1400D12B 00000000 T

```

Figure 15 More Simple Queries on bare PC

```

A6 : USERPROFILE/.sqliterc 0
SQLite version 3.7.6.2
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> create table t100(c1 int, c2 int, c3 char(20), c4 char(10), c5 int);
sqlite> insert into t100(1111, 2222, 'Karne', 'Okafor', 1000);
Error: near "1111": syntax error
sqlite> insert into t100 values(1111, 2222, 'Karne', 'Okafor', 1000);
sqlite> insert into t100 values(2222, 3333, 'Wang', 'Peter', 2000);
sqlite> insert into t100 values(3333, 4444, 'Mike', 'Wijesin', 3000);
sqlite> insert into t100 values(4444, 5555, 'Borg', 'Taylor', 4000);
sqlite> create table t200(c200 int, c201 char(30));
sqlite> insert into t200(8888, 'Table2');
Error: near "8888": syntax error
sqlite> insert into t200 values(8888, 'Table2');
sqlite> insert into t100 values(5555, 6666, 'Eyer', 'Matyas', 5000);
sqlite> select * from t100;
1111|2222|Karne|Okafor|1000
2222|3333|Wang|Peter|2000
3333|4444|Mike|Wijesin|3000
4444|5555|Borg|Taylor|4000
5555|6666|Eyer|Matyas|5000
sqlite> _

```

Figure 16 same queries on Visual Studio as in Fig 15


```

01 1          SQLITE Trasformation, Towson University          00000009
02 1          2          3          4          5          6          7          8
03 create table employee(name char(15), ssn int, addr char(20), dno int);
04 create table dept(dname char(10), mssn int, dnum int);
05 insert into employee values('Smith', 1111, 'MD', 1);
06 insert into employee values ('Wang', 2222, 'VA', 2);
07 insert into employee values('Okafor', 3333, 'NJ', 2);
08 insert into employee values('Lee', 4444, 'NY', 2);
09 insert into dept values('Research', 1111, 1);
10 insert into dept values('Admin', 2222, 2);
11 select name, addr from employee, dept where dname='Admin' and dnum=dno;
12
13 sqlite> _

15 name      addr
16 Lee       NY
17 Okafor    NJ
18 Wang      VA
19
20
21
22          dept      employee
23 0          00000000      1
24 00000000      C2 D2 E2          00000000 1400D12F 00000000

```

Figure 17 Bare PC Output Display of more complex queries QQ2

```

A6 : USERPROFILE/.sqliterc 0
SQLite version 3.7.6.2
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> create table employee(name char(15), ssn int, addr char(20), dno int);
sqlite> create table dept(dname char(10), mssn int, dnum int);
sqlite> insert into employee values ('Wang', 2222, 'VA', 2);
sqlite> insert into employee values ('Okafor', 3333, 'NJ', 2);
sqlite> insert into employee values ('Lee', 4444, 'NY', 2);
sqlite> insert into dept values ('Research', 1111, 1);
sqlite> insert into dept values ('Admin', 2222, 2);
sqlite> select name, addr from employee, dept where dname = 'Admin' and dnum = dno;
Lee|NY
Okafor|NJ
Wang|VA
sqlite>

```

Figure 18 same complex queries QQ2 on Visual Studio.

Chapter 9 RELATED WORKS.

Earlier discussions in section 5 have already identified BMC paradigm as the foundation 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. Several of them have been built and shown to outperform their OS-based counterparts [1, 3, 7, 10, 23]. With these, full BMC guidelines apply and there is a complete elimination of any underlying OS or middleware.

Another category of applications uses approaches that merely reduce OS overhead and/or use lean kernels. These include Exokernel [8], IO-Lite [22] and Palacios and Kitten[18]. With this category, there is still some OS presence, hence some difference with the BMC approach which represents complete elimination of underlying OS or centralized middleware.

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 ARM and Alfa binaries with reasonable code densities and quality [11]. In [24] 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 [4] 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 [2] 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.

BMC application development process has been described above in chapter 5; previous work on BMC and its characteristics are described in greater detail in [16].

For the purposes of transformation, application software is modeled simply as code that undergoes resolution of system calls/libraries during the compilation/link process so that appropriate bare machine interfaces to the hardware can be included with the application itself. Currently, this process is done manually as illustrated in this dissertation. Developing tools for automated transformation is the target of the next phase of the research on transformation; it is expected that such tools can transform a variety of applications to run on bare machines.

In [6], there is a description of how to port SQLite to new operating systems or how to create custom builds of SQLite.

In [25], a model to analyze tradeoffs between feature-rich and minimalist lean or “barebones” is presented as a cost/benefit analysis.

More and more transformation solutions are being offered on the web – some to accelerate growth in the enterprise, some to enhance the value of the organizations and some to help to drive growth and innovation; there are product offerings dealing also with transformations for different purposes. But in the end, we have not yet found any related research directly dealing with code transformation of an OS based application to a bare

PC application. This dissertation provides a foundation to transform OS applications to a bare PC or a bare system, in general.

Chapter 10 SIGNIFICANT CONTRIBUTIONS

Transforming SQLITE DBMS application to bare PC poses daunting challenges. This dissertation addressed many design issues and paved the way to further research in transformation process, where other applications can be transformed to run on bare machines. Some significant contributions of this research are outlined as follows:

- It discovered Visual Studio environment to be used as a first step in transformation process (during this step 1, it replaces as many system calls/libraries as possible with bare PC calls)
- In the second step of transformation a bare PC environment is used to transform the rest of the code from step 1 (step 2 transformation process is much smaller than step 1)
- Transforming SQLITE DBMS to run on a bare PC allows users to run their DBMS without having any OS on their computer (they can simply plug-in a detachable storage such as a USB and run their database system on a bare PC starting with a boot)
- It provided a standard methodology to transform an OS based application to a bare PC application
- It has blazed a trail for the transformation of other applications.
- A foundation is laid for the construction of tools that automate transformation (this dissertation demonstrated a manual version)
- It takes the code porting concept to the extreme level, where once a code is transformed, there may not be any need to port code from one platform to another

- It takes the ubiquity concept to the extreme level, where transformed code is ubiquitous without having a need for a virtual machine (like JVM)
- The direct hardware interfaces used to transform to bare PC code provide a new avenue, that leads towards building these interfaces into the hardware, thus making applications able to communicate directly with the hardware – and without any middleware
- When programs are made independent of OS or other computing environments, the resulting programs can be saved as archive files for the future where they can run on compatible CPU architectures, without a need for OS
- The transformation approach laid a ground work for future research in transformation of other programs written in other programming languages (instead of C/C++).

Chapter 11 SUMMARY

This doctoral dissertation presented a novel approach to transformation and a demonstration of a transformed SQLITE application that runs on a bare PC. Visual Studio platform was discovered as a medium that can be used to pseudo-transform code from Windows to bare PC applications, as was described fully in section 7. The pseudo transformation performed most of the transformation needed for bare PC environment.

The transformation process did not require any understanding or modification of the existing source code other than removing the **#include** files. As a way to check our results, the same sample queries are performed on VS and bare PC; the results from the transformed application are compared and validated against those from Visual Studio to make sure functionalities have been preserved. The successful transformation demonstrated in this dissertation offers potential means to transform other complex applications and systems.

The research presented here opens new avenues for making existing code independent of any operating systems or environments. The manual process used in transformation was in itself proof of concept; it also provides a methodology to construct a transformation tool for C/C++ or other programming applications.

This doctoral dissertation constitutes a trail blazer and serves as a cornerstone for future application transformations and related tools.

APPENDICES

APPENDIX A: SQLITE RESOURCES

1.Download link:

<http://www.sqlite.org/download.html>

This is the link for the source code downloaded for this research. It provides the ZIP archive containing all C source code for SQLite 3.7.16.1 thrown together into a single source file, usually referred to as the amalgamation.

In general, any information that is relevant to this research will be found by starting with the SQLITE Home page itself: <http://www.sqlite.org>.

2.Amalgamation package (available in [28])

This package comprises:

shell.c (main source file)

sqlite3.c (database source file)

sqlite3.h (header file)

sqlite3ext.h (header file)

3. User Interface

The SQLITE DBMS transformed in this dissertation is based on Windows platform. This package offers a user window interface, where SQL commands can be typed in and the results are displayed in the window. This package supports most of the standard SQL commands.

APPENDIX B: THE TWO DEVELOPMENT PLATFORMS

1. Visual Studio Development Environment

Visual Studio Development environment consists of a desktop PC with Windows XP and VS 10. The header files from the test environment are removed one at a time until most of the files are removed. NODEFAULTLIB option is used along with user libraries to create a test environment for removing the header files.

2. Bare PC Development Environment

Bare PC development environment uses Visual Studio 10 /bin directory only and does not use any /lib or /include files. The /bin directory provides the compilation and linking facilities. Compilation and linking in the bare PC environment is done using batch files which are simple and easy to use without depending upon any development IDE. The assembly files are compiled using asm.bat file. The C files are compiled using cpp.bat file. The programs are linked using ln.bat file. The mk.bat file provides means to create a bootable USB along with the application. Below follows a brief description of the critical files used in the bare PC platform.

1.asm.bat file listing is as follows:

```
rem asm.bat file  
  
..\bin\ml /c /Cx /Fl asmfiles.asm asmfilesb.asm chkstk.asm
```

Figure 19 – asm.bat file

The asmfiles.asm contains all the direct hardware interfaces required for most of the bare PC applications. The assembly calls are made through a C function call.

asmfilesb.asm file contains the specific direct hardware interfaces needed for SQLITE DBMS. The chkstk.asm is needed to provide a bare PC interface for stack checking.

2.cpp.bat file listing is as follows:

```
rem cpp.bat file
cls
rem needs microsoft visual c++ environment.
erase *.obj
erase *.*~
erase *.lst
erase shell.exe
call asm.bat
..\bin\cl /c /FA /ZI /Os /Gy /GS- /F64000000 shell.c sqlite3.c cfiles.c aaa.c
aaab.c memobj.c
```

Figure 20 – cpp.bat file

The shell.c and sqlite3.c are the two source files needed for SQLITE DBMS. The cfiles.c and aaa.c provide generic bare PC hardware interfaces through C functions. The aaab.c file provides specific bare PC hardware interfaces to SQLITE DBMS through C functions. The memobj.c file provides memory interfaces that are unique to bare PC environment. The bare PC applications use real memory and they manage their own memory map. The same memory object is also used in VS environment during the transformation process. However, in VS environment, one single malloc()

is used to obtain a large memory needed for SQLITE DBMS. This large memory is used to allocate memory needed within the SQLITE DBMS application.

3.ln.bat file is as follows:

```
rem ln.bat

cls

rem Needs Microsoft Incremental Linker (Microsoft Visual C++ Linker)

rem base address is 0x00100000

..\bin\link /MAP /BASE:0x00100000 /NODEFAULTLIB /OPT:NOREF

/MERGE:.rdata=.data /OPT:NOICF /STACK:32000000

/ENTRY:main shell.obj sqlite3.obj cfiles.obj aaa.obj aaab.obj asmfiles.obj

asmfilesb.obj memobj.obj chkstk.obj dir shell.exe
```

Figure 21 – ln.bat file

All bare PC applications use their own loader to load programs in memory. The BASE address is predefined in bare PC applications. The SQLITE DBMS is loaded at 0x00100000. However, a compiler always starts the starting address of the code at 0x1000. The real starting address in memory is thus 0x01001000. The NODEFAULTLIB options are used to completely eliminate OS libraries.

4.mk.bat file listing is as follows:

```
rem mk.bat file

format e: /FS:FAT32 /q

copy prcycle.exe e:

copy shell.exe e:

copy data.txt e:

rwhd -m 5 newbootf32.bin
```

Figure 22 – mk.bat file

The mk.bat file provides means to generate a bootable USB with boot, application and data. Initially, the USB flash drive (e.g. which is located in E drive) is formatted with FAT32 format. The pcycle.exe and shell.exe are copied to the USB. The pcycle.exe contains a menu program that facilities a user to load and run programs. This program provides all boot, IDT, GDT and TSS code in assembly. This code is written in TASM. The shell.exe is the SQLITE DBMS application single module. The data.txt file used to get any user data for the application. The rwhd.exe is used to create a bootable disk. This program is written in VS C++ and it does not depend upon any other boot code programs. All batch files described here run in a DOS window. All the development is done on a Windows Desktop.

3.How to get a list of system/library calls

The transformation process, in general, requires removing all the header files in the source files to make it compile for a bare PC environment. Some header files may

be independent of OS, but they contain all the definitions that may or may not be needed for a given application set. Thus, we remove all header files and substitute with header files that are only used in the application set and also needed for transforming to a bare PC application. The transformation process described in this dissertation clearly illustrated this process. However, when a new application is given for transformation, one need to immediately find the system/library calls needed which may or may not be available in the bare PC interfaces. These calls can be simply found by compiling the code and linking the code using NODEFAULTLIB option. In order to compile the code, you may need to provide some header files that contain some definitions and data types and structures. One can also use a VS environment to find the system/library calls by simply linking with NODEFAULTLIB option.

The following Fig.23 shows a list of *some* (we have shown only as many as can be contained in a simple text box) of the system/library calls for SQLITE DBMS after the SQLITE DBMS was successfully transformed to run on a bare PC.

```

[c:\home\student\UZO\sqliteuzobV81\interfaces]
CSBMC01$ cls

[c:\home\student\UZO\sqliteuzobV81\interfaces]
CSBMC01$ rem Needs Microsoft Incremental Linker (Microsoft Visual C++ Linker)

[c:\home\student\UZO\sqliteuzobV81\interfaces]
CSBMC01$ rem To link in 32-bit we need to use the above said linker

[c:\home\student\UZO\sqliteuzobV81\interfaces]
CSBMC01$ rem Run the Batch file MSDN.BAT

[c:\home\student\UZO\sqliteuzobV81\interfaces]
CSBMC01$ rem base address is 0x00100000

[c:\home\student\UZO\sqliteuzobV81\interfaces]
CSBMC01$ ..\bin\link /MAP /BASE:0x00100000 /NODEFAULTLIB /OPT:NOREF /MERGE:.rdata=.data /OPT:NOICF /STACK:32000000
/LIBPATH:"rkkvs.lib" /ENTRY:main shell.obj sqlite3.obj
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.

shell.obj : error LNK2019: unresolved external symbol _AOAfree referenced in function _main
sqlite3.obj : error LNK2001: unresolved external symbol _AOAfree
shell.obj : error LNK2019: unresolved external symbol _AOAmalloc referenced in function _main
sqlite3.obj : error LNK2001: unresolved external symbol _AOAmalloc
shell.obj : error LNK2019: unresolved external symbol _printf referenced in function _main
sqlite3.obj : error LNK2001: unresolved external symbol _printf
shell.obj : error LNK2019: unresolved external symbol _AOAmemcpy referenced in function _main
sqlite3.obj : error LNK2001: unresolved external symbol _AOAmemcpy
shell.obj : error LNK2019: unresolved external symbol _AOAPrintText referenced in function _main
shell.obj : error LNK2019: unresolved external symbol __AOAaccess referenced in function _main
_main
shell.obj : error LNK2019: unresolved external symbol _AOAmeminit referenced in function _main
shell.obj : error LNK2019: unresolved external symbol __AOAisatty referenced in function _main
shell.obj : error LNK2019: unresolved external symbol __allmul referenced in function _main
sqlite3.obj : error LNK2001: unresolved external symbol __allmul
shell.obj : error LNK2019: unresolved external symbol _AOAisdigit referenced in function _isNumber
shell.obj : error LNK2019: unresolved external symbol _AOAisprint referenced in function _output_c_string
shell.obj : error LNK2019: unresolved external symbol _fputc referenced in function _output_c_string
shell.obj : error LNK2019: unresolved external symbol _putc referenced in function _output_csv
shell.obj : error LNK2019: unresolved external symbol _fclose referenced in function _do_meta_command
shell.obj : error LNK2019: unresolved external symbol _fopen referenced in function _do_meta_command
shell.obj : error LNK2019: unresolved external symbol _AOAmemset referenced in function _do_meta_command
sqlite3.obj : error LNK2001: unresolved external symbol _AOAmemset
shell.obj : error LNK2019: unresolved external symbol _AOAstrncmp referenced in function _do_meta_command
referenced in function _seekWinFile
_winShmSystemLock

sqlite3.obj : error LNK2019: unresolved external symbol _AOAGetFullPathNameA referenced in function _winFullPathname
sqlite3.obj : error LNK2019: unresolved external symbol _AOAGetFullPathNameW referenced in function _
[c:\home\student\UZO\sqliteuzobV81\interfaces]
CSBMC01$ dir shell.exe
Volume in drive C has no label.
Volume Serial Number is 725A-AECD

Directory of c:\home\student\UZO\sqliteuzobV81\interfaces

```

Figure 23 – System Calls from using NODEFAULTLIB option.

4.Sample bare PC direct hardware interfaces – Memory Objects

```

/*****
/* memobj.c file name */
/* coded by Dr. Ramesh K. Karne */
/* November 29, 2012 */
/* Towson University */
/* BMC LAB */
*****/
#include "memobj.h"

void AOAfree(void *ptr);
void * AOAmalloc(unsigned int size);
void * AOArealloc(void *ptr, unsigned int size);
void AOAmemint (int tablebase, int membase);

//-----
// memory object initialization
//-----

void AOAmeminit(int tablebase, int membase)
{
    int i = 0;

    mem_alloc_table_base = (int*)tablebase;
    global_mem_addr_base = (char *)membase;

    for (i=0; i < MAX_MEM_ALLOC_TABLE_SIZE; i++)
    {
        mem_alloc_table_base[i] = 0; //reset table
    }
    for (i=0; i < GLOBAL_MEM_SIZE; i++)
    {
        global_mem_addr_base[i] = 0; //reset dynamic memory
    }
    mem_alloc_table_ptr = (int*)tablebase; //initialize ptr
    global_mem_addr_ptr = (char*)membase;

};
//-----

```

Figure 24 bare PC memory object - Initialization


```

//AOAmalloc()
//-----
void * AOAmalloc(unsigned int size)
{
    char *ptr;
    int i=0;
    int tablesize;
    unsigned int memsize;
    unsigned int entry;
    int *table;

    if (size > 4096*4096) //16MB limit
    {
        printf("Memory allocation request exceeds limits: %x \n", size);
        return (void*) -1;
    }

    memsize = size + global_memory_size;
    if (memsize > GLOBAL_MEM_SIZE) //16MB limit
    {
        printf("Memory allocation failed, ran out of memory: %x \n", memsize);
        return(void*) -2;
    }
    tablesize = 8 + mem_alloc_table_size;
    if (tablesize > MAX_MEM_ALLOC_TABLE_SIZE*4) //1000 entries
    {
        printf("Memory allocation failed, table is full: %x \n", tablesize);
        return(void*) -3;
    }
}

```

Figure 25 bare PC memory object – Allocate

```

entry = 0;
table = (int*)mem_alloc_table_ptr; //current pointer for table
ptr = (char*)global_mem_addr_ptr; //current pointer for allocated memory

entry = (unsigned int)ptr;          //current address
*(table + 1) = entry;               //store the allocated address
entry = size;
entry = entry << 8;                 //shift address by one byte
entry = entry + 1;                 //add valid bit for the entry
*table = entry;                     //store the record entry
mem_alloc_table_ptr = mem_alloc_table_ptr + 8; //increment the pointer
global_mem_addr_ptr = global_mem_addr_ptr + size; //add the current size

if (ptr >= (char*) 0x28000000)
{
    AOAPrintText("Memory problem", 3800);
    AOAEExit();
}

//printf("size: %x memptr: %x tableptr0: %x tablepointer1: %x \n", size, ptr, *table,
*(table+1));

//scanf_s("%d", &i);
return ptr;
};
//-----

```

Figure 26 bare PC memory object – Allocate continued

```

//Memory Object Function
// free memory
//-----
void AOAFree(void *ptr)
{
    int flag = 0;
    int i=0;
    unsigned int entry;
    unsigned int addr;
    unsigned int addrtable;
    int *table;    //uzo
    int index;
    int valid = 0;

    return;

    entry = 0;
    addr = (unsigned int)ptr; //address to be free

    table = (int*)mem_alloc_table_base;    //base of the table
    //printf("table base: %x ptr: %x \n", table, ptr);
    index = 0;
    while (flag == 0 && index < MAX_MEM_ALLOC_TABLE_SIZE*2)
    {
        addrtable = *(table+1); //get the address in the table
        valid = *table & 0x000000ff; //extract valid byte
        if (valid == 0x01 & addrtable == addr)
        {
            //entry found in the table
            *(table+1) = 0;
            *table = 0;
            flag = 1;
            break;
        }
        index++;
        table = table + 8;    //next entry
    }
    printf ("free entry: %x address released: %x \n", index, ptr);

    return;
}

```

Figure 27 bare PC memory object – Free memory

```

//-----
//MEMORY OBJECT realloc()
//-----
void * AOArealloc(void *ptr, unsigned int size)
{
    int flag = 0;
    int i=0;
    unsigned int entry;
    unsigned int addr;
    unsigned int addrtable;
    int *table;    //uzo
    int index;
    int valid = 0;
    unsigned int oldsize = 0;
    unsigned int lesssize = 0;
    int count = 0;
    char *nptr;
    char *optr;

    entry = 0;
    addr = (unsigned int)ptr; //address to be reallocated
    table = (int*)mem_alloc_table_base;    //base of the table

    //printf("table base: %x ptr: %x \n", table, ptr);

    index = 0;
    if (ptr == 0)
    {
        return AOAmalloc(size); //null pointer, do a new malloc and get the new address
    }
    if (size == 0)
    {
        return 0;
    }

    while (flag == 0 && index < MAX_MEM_ALLOC_TABLE_SIZE)
    {
        addrtable = *(table+1); //get the address in the table
        valid = *table & 0x000000ff; //extract valid byte
    }

```

Figure 28 bare PC memory object – Reallocate

```

        if (valid == 0x01 & addrtable == addr)
        {
            oldsize = *table & 0xfffff00; //remove valid byte
            oldsize = oldsize >> 8;    //extract old size from the record
            optr = (char*)addrtable;    //old memory ptr
            if (oldsize >= size)
                lesssize = size;    //capture less size
            else
                lesssize = oldsize;
            nptr = AOAmalloc(size); //get a new address, new memory ptr
            for (i=0; i<lesssize; i++)
            {
                nptr[i] = optr[i]; //restore memory in new memory
            }
            flag = 1;
            break;
        }
        index++;
        table = table + 8;    //next entry
    }

    //printf("newsize: %x oldsize: %x lesssize: %x optr: %x nptr: %x \n", size, oldsize,
    lesssize, optr, nptr);

    return (void*) nptr;
};

```

Figure 29 bare PC memory object – Reallocate Continued

REFERENCES

- [1] A. Alexander, R. Yasinovskyy, A. L. Wijesinha, and R. K. Karne, "SIP Server Implementation and Performance on a Bare PC," International Journal in Advances on Telecommunications, vol. 4, no. 1 and 2, 2011.
- [2] W. Ahn, S. Qi, M. Nicolaides, and J. Torrellas, "BulkCompiler: High-Performance Sequential Consistency through Cooperative Compiler and Hardware Support," IEEE/ACM International Symposium on Micro Architecture, 2009.
- [3] P. Appiah-Kubi, R. K. Karne and A.L. Wijesinha, "The Design and Performance of a Bare PC Webmail Server," The 12th IEEE International Conference on High Performance Computing and Communications, AHPCC, pp. 521-526, Sept. 2010.
- [4] R. Cáceres, C. Carter, C. Narayanaswami, and M. Raghunath, "Reincarnating PCs with Portable SoulPads", IBM T.J. Watson Research Center, New York
- [5] C Language Library, <http://www.cplusplus.com/reference/clibrary/>.
- [6] Custom Builds Of SQLite or Porting SQLite To New Operating Systems, <http://www.sqlite.org/custombuild.html>.
- [7] P. A. Emdadi, R. K. Karne, and A. L. Wijesinha. Implementing the TLS Protocol on a Bare PC, ICCRD2010, The 2nd International Conference on Computer Research and Development, Kaula Lumpur, Malaysia, May 2010
- [8] D. Engler, "The Exokernel Operating Systems Architecture," Dept. of Elec. Eng. and Computer Science, Massachusetts Institute of Technology, Ph.D. Dissertation, 1998.
- [9] FreeBSD/Linux Kernel Cross Reference, http://fxr.watson.org/fxr/source/kern/syscalls.c_

- [10]L. He, R. K. Karne, and A. L. Wijesinha, "Design and Performance of a bare PC Web Server," International Journal of Computer and Applications, vol. 15, pp. 100-112, June 2008.
- [11]Y. Hwang, T Lin., R. Chang, "**DisIRer**: Converting a Retargetable Compiler into a Multiplatform Binary Translator. In ACM Transactions on Architecture and Code Optimization," vol. 7, issue 4. 2010.
- [12]R. K. Karne, "Application-oriented Object Architecture: A Revolutionary Approach," 6th International Conference, HPC Asia, Dec. 2002.
- [13]R. K. Karne, S. Liang, A. L. Wijesinha and P. Appiah-Kubi, "Bare PC Mass Storage USB Driver," International Journal of Computer and Applications, March 2013.
- [14]R. K. Karne, V. Jaganathan, T. Ahmed and N. Rosa, "DOSC: Dispersed Operating System Computing," OOPSLA, Onward Track, 20th Annual ACM Conference on Object Oriented Programming, Oct. 2005.
- [15]R. K. Karne, V. Jaganathan and T. Ahmed, "How to run C++ Applications on a bare PC," 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD), pp. 50-55, May 2005.
- [16]G. H. Khaksari, A. L. Wijesinha, and R. K. Karne, "A Bare Machine Development Methodology," International Journal of Computer Applications, vol. 19, no.1, pp. 10-25, Mar. 2012
- [17]G. H. Khaksari, A. L., Wijesinha, R. K., Karne, L., He and S. Girumala, "A Peer-to-Peer Bare PC VoIP Application," IEEE Consumer and Communications and Networking Conference (CCNC), pp. 803-807, Jan. 2007.

- [18]J. Lange. et. al, “Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing,” 24th IEEE International Parallel and Distributed Processing Symposium , Apr. 2010.
- [19]U. Okafor, R. K. Karne, A. L. Wijesinha and B. S. Rawal, "Transforming SQLITE to run on a bare PC," ICSOFT, 2012.
- [20]U. Okafor, R. K. Karne, A. L. Wijesinha and P. Appiah-Kubi, "Eliminating The Operating System via the Bare Machine Computing Paradigm," IARIA, 2013.
- [21]U. Okafor, R. K. Karne, A. L. Wijesinha and P. Appiah-Kubi, "A Methodology to Transform an OS-based Application to a Bare Machine Application," IUCC, 2013.
- [22]V. S. Pai, P. Druschel and Zwaenepoel, “IO-Lite: A Unified I/O Buffering and Caching System,” ACM Transactions on Computer Systems, vol.18 (1), pp. 37-66, Feb. 2000.
- [23]B. Rawal, R. K.Karne, A. L. Wijesinha, “Mini Web Server Clusters for HTTP Request Splitting,” IEEE International Conference on High Performance Computing and Communications, pp. 94-100, Sep. 2011.
- [24]M. Schoeberl, S. Korsholm, T. Kalibera and A. P. Ravn, “A Hardware Abstraction Layer in Java,” ACM Transactions on Embedded Computing Systems, vol.10, no. 4, Article 42, Nov. 2011.
- [25]S. Soumya, R. Guerin and K. Hosanagar, “Functionality-rich vs Minimalist Platforms: A Two-sided Market Analysis”, ACM Computer Communication Review, vol. 41, no. 5, pp. 36-43, Sep. 2011.
- [26]SQLITE Download: <http://www.sqlite.org/download.html>, [sqlite-amalgamation-3071000.zip](#).

[27]Windows System Call Table, Googlecode.com, retrieved Feb16,2012,
<http://miscellaneous.googlecode.com/svn/trunk/winsyscalls.html>

[28]www.sqlite.org

CURRICULUM VITA.

Uzo Okafor

1573 Ingleside Ave,

Gwynn Oaks MD 21207

973-332-0997(cell).

email: uo07041@yahoo.com

OBJECTIVE: Lead Developer, Technical Project Manager or Branch Chief in an area requiring use of skills in both mainframe and Web processing and/or a position as a Professor of Computer Science, Research in mainframe and WebSphere applications.

SKILLS: Application Systems Analysis, Design and implementation on MVS, IMS DB/DC, CICS command level programming, COBOL2, JCL, TSO/SPF, Panvalet, Librarian, DB2, Excelerator, Microsoft Windows software (NT, Word, Access, Excel, Exchange), MicroFocus Cobol workbench/XDB and familiarity with PowerBuilder, Object Oriented Design. Object Oriented Programming in C++, JAVA, Oracle8, Visual Basic.

EDUCATION:

2009 – 2013: D.Sc, Information Technology, Towson University. Towson, MD.

02/2000 - 08/2000: Complete-Pro course Certification at Comp-u-learn, Edison NJ.

Programming concepts in C and object-oriented concepts in C++, design, develop

and program relational databases using Oracle8, SQL and PL/SQL, Design and develop Web pages using HTML, develop Windows and Internet applications using JAVA and Visual Basic.

1974 – 1976: M.S in Computer Science, Indiana University, Bloomington, Indiana. Earned tuition and living expenses through departmental teaching assistantship (tuition waiver + stipend). Basic courses studied included Operating systems, Systems Design and implementation, Numerical Analysis, Computational logic, Automata theory, Formal grammars, Optimization techniques.

1971 – 1974: B.Sc. (Upper 2nd class Honors) in Pure & Applied Mathematics, University of Dar-es-salaam, Tanzania. Was sponsored through the African Universities scholarship program (AFGRAD equivalent at the undergraduate level)

PUBLICATIONS

- (1) Uzo Okafor, Ramesh K. Karne, Alexander L. Wijesinha, and Bharat S. Rawal, Transforming SQLITE to Run on a Bare PC, ICSOFT, page 311-314. SciTePress, (2012)
- (2) Uzo Okafor, Ramesh K. Karne, Alexander L. Wijesinha, and Patrick Appiah-Kubi, Eliminating the Operating System via the Bare Machine Computing Paradigm, IARIA 2013.
- (3) Uzo Okafor, Ramesh K. Karne, Alexander L. Wijesinha, and Patrick Appiah-Kubi, A Methodology to Transform an OS-based Application to a Bare Machine Application, Submitted

RESEARCH EXPERIENCE:

2010 – 2013: Doctoral Research: Bare Machine Computing on Applications.
SQLITE Transformation to run in a Bare Machine Computing Environment.

Department of Computer and Information Science

Towson University, Towson, MD

Advisor: Ramesh Karne Ph.D.

This research deals with transforming SQLITE DBMS system to run in an
environment without an operating system.

INDUSTRY EXPERIENCE:

April 2003 – Present: Information Technology Specialist, Social Security
Administration, Baltimore, Md. 21235.

Provide technical advice to managers of the Human Resource Department, create
alternative systems solutions to user requirements and needs, attend user meetings,
review and evaluate periodic systems needs/change requests of the department,
participate in subsequent Design meetings within the application teams, design
programs, code COBOL programs, perform unit testing and provide general technical
support to Integration Testing groups prior to implementation in production. Since
hiring into SSA, I have also completed several change requests in the Mainframe
Time and Attendance System (MTAS), the Human Resource Management
Information System (HRMIS/HRODS). Lately I work on both the RMTAS, SPARS
and TPPS projects which are supported by large DB2 databases. My work requires
strong skills in CICS COBOL, DB2 databases, SQL and VSAM file processing.

Jun 1994 – May 2002: Senior Technical Member of Staff, CSC/AT&T Account, Somerset, NJ RMMS application. Design meetings, designed programs, coded COBOL programs for several re-engineering projects in the Vendor Update Processing (VUP) subsystem. Was the leader for application development effort for several releases of the VUP subsystem, which provided new functionalities. Supervised the production support work of junior developers in the VUP team, led many investigations for clients. Assignments were completed in IMS DB/DC (batch and online), DB2, COBOL II on MVS and included writing program specs, physical program design, coding and Implementation. MicroFocus Cobol/XDB was used as a development tool. Was a member of the CSC team in the CSC/Federal sector which at the time completed a major outsourcing transition work with the Army - the WLMP/LOGMOD project.

1983 – 1994: was a level A5 Technical Member of Staff, Harris District, AT&T, East Brunswick in CCS/CIS organization. Was Project Leader of Common Tables group (CTS). As such was responsible for providing new application Systems and Production Support for the reference table data needed for AT&T's Residence Long Distance billing. Responsibilities as a Project Leader included receiving and fulfilling requirements from user community, overseeing analysis, design, coding and testing of software changes required, from the user groups, for each release. Was responsible for planning, directing and supervising the work of nine other developers, and continually interfacing with peers in other support groups such as DBA, CMCC Installation, Corporate ITS (Alpharetta and Orlando), coordinating critical interfaces between CTS and other AT&T sub-applications such as RAMP, IDB, RIPS and NPP.

1981 - 83: Senior Programmer/analyst Cameron Iron Works, Houston, Texas. I was assigned the dual responsibility of maintaining the old Sales Order system on MVS and the rewrite of the system to use an IMS database master file with real time online processing in Command Level CICS/VSAM file interface. Maintained old system with changes requested for current production submitted proposal for a new database, created a logical model for it, worked with a DBA group to create test physical data base. Coded programs for conversion to the new database file, wrote specifications for other programs required for new project. Trained new hires for the project

HONORS AND AWARDS:

1971 - 74 Sponsored through the African Universities scholarship program
(AFGRAD equivalent at the undergraduate level) for undergraduate studies at
the University of Dar es Salaam

1974 – 76 A Departmental teaching assistantship (tuition waiver + stipend) at Indiana
University, Bloomington, Indiana.

April 2004 - Social Security Administration: Commendable Act or Service
(CAS) award. This is a *Special Act or Service Award* which recognizes
individuals for major accomplishments or contributions, during the *current*
assessment cycle, that have promoted the mission of the organization

May 2005 – Social Security Administration: similar award to the above

October 2007 – Social Security Administration: Associate Commissioner's Citation is
an award for outstanding support to the Office of Personnel (OPE) in

maintaining the Human resources Management Information System (HRMIS) and Human Resources Operational Data Store.

March 2010 – Social Security Administration: Deputy Commissioner's Citation is an award for outstanding dedication and support in developing the Agency's new and improved web-based Performance Assessment & Communication System (PACS)

