Efficient Migration to Windows

Pallavi Kalyanasundaram[1*], Sunita P. Ugale[1], Smitha K. P.[2], Priti Ranadive[2]
[1] K.K.W.I.E.E.R., Nasik, Maharashtra, India.
[2] Crest, KPIT Technologies Ltd., Pune, Maharashtra, India.

* Corresponding author. Tel.: +91-8975899743; email: pallavi.k7491@gmail.com
Manuscript submitted February 15, 2015; accepted July 6, 2015.
doi: 10.17706/jsw.10.8.1030-1036

Abstract: Software portability is gaining importance worldwide as it adds value by increasing the shelf life of a software application. One aspect of portability deals with porting software across multiple operating systems. Amongst the available, Windows and Linux are widely used operating systems. There are various methods for porting a software tool or application developed in Windows to Linux, however, very less has been written about methods to port Linux applications to Windows. Our paper highlights some popular methods for porting a software application written in C from Linux to Windows. The technique emphasized in this paper provides a simple menu driven environment which assists a person completely unaware of Linux to port a C application efficiently to Windows. Our paper explores all aspects of how one can systematically port C application developed on Linux, gain access to C source code on Windows, handle issues related to standard header files, libraries, file translations, compiling, linking, debugging and distribution with the help of a case study. The case study addresses issues related to porting huge C application composed of multiple executable modules with each module involving multiple C files. The paper also highlights the Windows equivalents of Linux command line utilities or tools that may be required for functional verification and debugging.

Key words: Batch script, cygwin, microsoft visual C++ 2008 express edition, porting, software maintenance, software requirement engineering, static and dynamic libraries, virtual machines.

1. Introduction

Porting is the process of executing a software in an environment that is different from the one for which it is originally designed. The term environment refers to all the components that interact with the ported software. Environment may include software execution across different operating systems, processors or compilers.

Porting classifies under the umbrella of software requirements engineering or software maintenance. Before porting the software, the overheads required for porting must be assessed. Overheads may typically include time, cost and benefit. Based on these overheads, decision must be taken to either go for porting or rewriting the whole code for the intended platform. Complexities involved in porting depend upon the application which has to be ported. [1] finds that developers often make mistakes while porting code and these errors take about a year to be detected and fixed on average. [2] gives an overall picture of the porting process considerations from a business perspective.

Source code porting is of great significance as it increases the range of environments on which the software can be used in comparison to porting binaries which requires highly similar platforms. The

process of porting the source code involves the following steps:

1) Gaining access to source codes on the intended platform
2) Examining make files, header files, and source code for porting issues
3) Compiling and Linking
4) Fixing Linkage issues
5) Debugging
6) Handling Scripts
7) Integration and Distribution of application [3]

Large number applications are written in C which is one of the widely used programming languages. Similarly, Linux OS has gained popularity due to open source licensing policy and more control over the OS. Thus, many of the C based applications are developed on Linux. As Linux environment offers command line interface which requires some proficiency, a Windows version of the software may serve as useful feature.

In this paper, we discuss a case study for porting a software tool written in C, originally built on Linux, to Windows environment using Microsoft Visual C++ 2008 Express Edition Integrated Development Environment (IDE). This method is attractive because the person need not have background knowledge of Linux. Section 2 mentions the current work and trends in the porting domain. Section 3 elaborates the case study with all the issues/errors that occurred during actual porting of the application and how they are handled. Section 3 also describes the Linux tools or commands used for functional verification of the software along with its Windows equivalents which may serve as prerequisites while porting an application.

2. Literature Survey

If the cost of porting a program to another OS is more than writing a different version of it for that OS, the program isn't portable [4]. Degree to which software is made portable plays an important role in saving time, money and efforts. Portable software sustains in the market for a longer time but it requires skilled and experienced developers. Current research in the field of porting deals with creating automated software tools to analyze the code for various aspects of porting.

This paragraph reviews the contributions of [1]. It categorizes porting errors related to inconsistent control flow, inconsistent identifier renaming, inconsistent data flow, and code redundancy based on the porting errors and fixes reported by developers. The database comprises of 113 and 182 porting errors retrieved from the version histories of FreeBSD and Linux, respectively. [1] implements REPERTOIRE – a cross platform porting analysis tool which computes the extent of cross-system porting between two patches. Patches here imply that piece of code which developers usually copy to reuse or modify functionality. In a large industrial product line, about 4% of the bugs are duplicated across more than one product or file [5]. REPERTOIRE will aid to understand the time and effort required for porting the code and which part of the code is more prone to porting errors. It also proposes SPA tool to detect semantic porting inconsistencies.

There are various emulators available for applications developed on Linux to be executed on Windows viz., Cygwin, Virtual Machine and various others. Cygwin is a set of open source tools which provides Unix-like environment and assists to port applications from UNIX/Linux to the Windows environment. To use Cygwin, knowledge of Linux is mandatory. [6] is a review article that examines the Cygwin toolkit. The second method is using VMWare (Virtual Machine Software) with which one can operate another OS on a computer with separated desktop, hard drives and everything. [7] mentions the state-of-the-art of virtualization with its benefits and drawbacks.

In this paper, we have ported a Linux based C application to Microsoft Visual C++ 2008 Express Edition

Integrated Development Environment (IDE). The first two methods have their own advantages but the IDE method opted here offers scope for optimization on Windows which is not possible using other two methods. Some other features of the IDE include simple menu interface which takes less time to understand, easy to use, good Graphical User Interface (GUI) [8] which is explored in the following section.

3. Case Study

We elaborate the process by porting a C application composed of 1) Frontend - part of compiler comprising of four modules – preprocessor, post-preprocessor, scanner – parser and MXML library. Frontend generates XML file as output. 2) Backend - an application code which uses the XML file generated by frontend.

Fig. 1 shows the ported C Application. The application comprises of five modules. The first module is an open source MCPP preprocessor available for Windows as well as Linux. By enabling macro for the Windows version in the MCPP source code, the preprocessor Module 1 is ported without additional efforts. Module 2 and Module 3 are C codes handled for similar set of porting issues discussed next. Module 4 is MXML library, an open source C library. The library files on Linux have .a extension which is incompatible with the Windows IDE. So, the MXML library is compiled on the VC++ IDE to generate a .lib file instead of an executable. Modules 1 - 4 form the frontend of the application which generates XML file as output. This XML file is used by Module 5 which is a C application considered for porting issues. Rest of the paper walks through the issues and solutions involved at various stages of porting this software tool successfully from Linux to Windows.

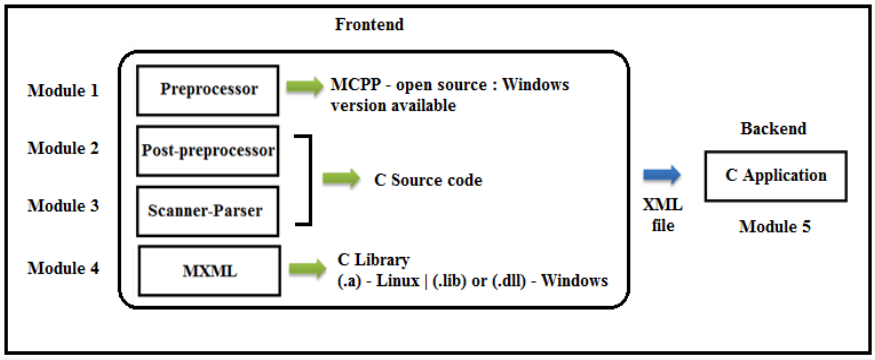

Fig. 1. Ported C application.

Each module of the application is executed individually and the same process is repeated for each module. Modular approach is systematic and reduces the overall complexity of a huge code set which needs to be ported. This accelerates the process of task completion. Once the Windows counterpart for each Linux executable module is created, the final integration is done using Batch script.

First we obtain the Linux source code on Windows Microsoft Visual C++ IDE. After adding all the C and header files in the project directory of the IDE, the next step is to handle header files and resolve the problems associated with it. Functionality of each module is verified by comparing the output files generated by each module on Windows with its Linux equivalent. The comparison methods are discussed in sub-section 3.5.

3.1. Handling Header Files

Standard and user defined header files on Linux and Windows need to be handled properly to avoid porting errors. Errors observed during porting header files and their respective solution is described next.

1) **Directory of user defined header files:** In VC++ IDE, user defined header files should be inside the

current project directory. On Linux, the user defined header files can be in a different directory than the current working directory.

Example: On Linux: #include "../user_defined_header.h" is possible

Above example shows that the user_defined_header.h is located in the outer directory of the current working directory. The Windows IDE does not support such folder structures. The source code must be analyzed for such header file paths during porting.

2) **Difference between user defined and standard header files:** User defined header files #include"user_header.h" and **standard** header files & #include<stdio.h> are strictly represented differently. The double inverted commas and angular brackets of the respective header files should be taken care of.
3) **Linux standard files:** Standard Linux header files like unistd.h cannot be used on Windows platform directly. The source code **should** be analyzed for Linux based functions. Equivalent windows function should be found or separate code should be written to replicate the functionality.
4) **Similar header files on Linux and Windows:** Care must be taken while using header file equivalents e.g. sys/time.h on Linux and time.h in Windows – all functionalities are not exactly same, the function definitions may be different.
5) **Application specific issue:** The code demands to parse the header files. The header files definitions and keywords on Linux and Windows are different which results in parsing error during execution. There are two **different** approaches to avoid such errors, the first is to handle the keywords internally in the code and second is to handle the keywords externally. Elaborating more on the second approach, example, __int64 in stdio.h header file (Windows) is equivalent to long long datatype. Hence, the files which need stdio.h the keyword __int64 is replaced by long long. Note that the changes are not made in the standard header files, it is handled when the keywords are used in the application. There are various ways to replace the keywords which are mentioned in subsection 3.4.

3.2. Problems Associated with Compiling and Linking

Depending on the degree of portability of the source code, the count of compiling and linking errors will vary. List of compiler errors in the IDE are mentioned in [9]. Following mentioned are some typical errors. All the solutions are highlighted and the best out of them is used for implementation.

1) **Language standards:** Many compiling errors occur due to the Language standards like C90, C99, C11 not being met. In this context the errors may occur because some standards allow initializations only at the start of the blo**c**k code i.e. exactly after the opening curly brackets. Initializations at any random point are not allowed.

Example: Results in compile time error.

```
if (condition)
{
        Operation 1;
        Data_type variable_name = initial_value;
}
```

2) **Linking library files:** Issues occur while linking library files. One reason being .a extension of static library files (usually on Linux) is not supported on Windows. Using IDE approach, Static (.lib) or Dynamic (.dll) (dll stands for dynamic linking libraries) **library** files can be easily created. Maintenance and resource problems are associated with static libraries, so the modern systems today use Dynamic Link Libraries (DLLs) or shared libraries [10].
3) **Compiler directives:** Using '#warning' directive in the application code gives compile time error. Replacing it with '#pragma message' solves the problem. This issue is common with applications using

preprocessors.

4) **Space between path names:** Issues **arise** due to space in between path names causing linking errors.

3.3. Debugging

Logical errors are difficult to diagnose. The developer or programmer needs to consider the following points to avoid such scenarios.

1) **Line endings of files:** Line endings i.e. moving to the next line of the code is an important factor to be considered. Line Feed (LF) denoted by \n and Carriage Return (CR) denoted by \r are the two characters associated with line endings. On Linux, next line uses the character LF while on Windows next line is denoted by the character sequence (CR-LF). There are various methods to handle line endings. Convert or translate the text files from Linux to Windows by using unix2dos command and dos2unix command for vice versa situation.
2) File pointers and line endings: Line Endings create problem while reading files and dealing with file pointers. fseek command uses file pointers which results in wrong functionality as it counts two characters for every line ending instead of one. Solution is to open files in binary mode whenever file pointers are used.

Example: FILE * fopen (const char * filename, const char * mode);

The mode 'r' i.e. read mode indicates translated mode. It will translate the line endings according to the current platform. So, file pointers will point to incorrect location if a Linux file is opened in Windows. If the file is opened in 'b' mode i.e. binary mode it is a non-translated mode. It does not suppress the line endings according to the environment hence giving correct results. There are various other modes in which file can be opened. Every mode can be appended with 'b' which ensures that file is opened in non-translated mode.

Example: fopen ("filename", "rb");

3) **Tools to trace issues:** At the time of development, the programmer should ensure that memory leaks do not occur in the code. There are tools available to detect memory leakage. Valgrind – A Memory leakage detection tool, available for Linux and Windows. The Windows VC++ IDE has its own debugger which traces the point of memory leakage. Following the call stack, the exact cause of leakage can be identified and resolved. 'gdb' and 'ddd' are some commonly used debuggers on Linux Platform.

3.4. Other Issues

There are some additional issues which may occur and cannot be categorized in the above subsections.

1) **Using variables as keywords**: During debugging, **value** of the variable is displayed when the cursor is placed on variable. If keywords are used as variables the values cannot be observed but the logic is not affected.
2) **Handling keywords of Windows standard header files:** There is a requirement in the application to search and replace certain words in some files as mentioned in sub-section 3.1 point 5). To handle replacement, a batch script is written. Complete description about batch files and how it is used, including the syntax and errors can be found in [11]. A problem is observed in handling special characters in the files when implemented using batch scripts. <. >, & etc., all are special characters which are prominently used in C. As the whole file is searched for the particular key word, when the script comes across the special characters it produces garbage value. Code for handling special characters in the script must be added. Escape sequence '^' is used to handle special characters in batch script. This increases execution time of the script. Alternative **method** is to write a C code for replacing words in a file. Replace using batch commands may even cause logical errors i.e. compiling successfully but ending up in wrong functionality. Hence, writing a C code is much efficient in such situation.

3.5. Testing and Verification

As the application is executed module by module, the output files generated for each module are compared with its Linux counterpart. The corresponding files must be identical, except for some Windows specific keywords which may differ as the header file definitions are different. For comparison of two files, Linux uses "meld" command (other options available). The alternative file comparison tool on Windows can be the Compare Plugin in Notepad++.

3.6. Integration

After each executable module is verified for its correct functionality the next step is to integrate them. The integration and execution of these modules in a specific order leads to the successful porting of the overall software tool. There are several methods to integrate the modules.

1) The first method is to write a C code and use the system function 'system()' to execute each module. There are some security concerns related to using system() function hence it should be avoided. Moreover, it can accept only fixed command line arguments as its value cannot be updated.
2) The second method is to use Batch scripts described in subsection 3.4. Batch file (.bat) is executed on the command prompt using 'run' command. It is easy to integrate using this method.

3.7. Distribution

Once the Linux developed tool is working on the Windows system, we need to ensure that it works on all other Windows systems apart from the system on which it is actually developed. The pre-requisites in order to make the tool work on other machines, the system should have Redistributable VC++ library packages already installed. Moreover, the modules must be compiled in release mode of the IDE instead of debug mode.

4. Conclusion

The paper discusses the necessary points to be considered while porting a software application consisting of multiple C files from Linux to Windows. We mention the problems at various stages of porting and present the different methods to solve them and which solution is perfect for the problem. The paper covers some techniques to check the functionality of the software tool after porting. The paper highlights the porting process for a particular application. Depending on the application to be ported, complexities may be different and other problems may also be observed apart from those which are discussed. At the time of application development itself, the developer should consider all the aspects to make the application platform independent. It is observed that, if the application is modular, less time is required to port the same.

References

[1] Baishakhi, R. (2013). Analysis of cross-system porting and porting errors in Software projects. Appears in UT Electronic Theses and Dissertations, University of Texas, Austin.

[2] Alfredo, M., Chakarat, S., & Artis, W. (2006). *UNIX to Linux® Porting: A Comprehensive Reference*, Prentice Hall.

[3] Porting Steps. Retrieved from website: http://www.mkssoftware.com/solutions/porting.asp

[4] Ken, G. (2007, November). Software Portability: Weighing Options, Making Choices. Retrieved February 10, 2015, from http://www.nysscpa.org/cpajournal/2007/1107/perspectives/p10.htm

[5] Li, J. Y., & Michael, D. E. (2012). Cbcd: cloned buggy code detector. *Proceedings of the International Conference on Software Engineering*.

[6] Jeffrey, S. R. (2000). The Cygwin tools: A GNU toolkit for Windows. *Journal of Applied Econometrics*,

15(3).
[7] Fatma, B., Chan, Y. Y., & Mohamed, J. Z. (2012). State-of-the-art of virtualization, Its security threats and deployment models. *International Journal for Information Security Research*.
[8] Ivor, H. (2008). *Beginning VISUAL C++ 2008*. John Wiley & Sons.
[9] Visual Studio C/C++ Build Errors. Retrieved from http://msdn.microsoft.com/en-us/library/8x5x43k7.aspx
[10] David M. B., Brian, D. W., & Ian, R. C. (2001). The inside story on shared libraries and dynamic loading. *Computing in Science and Engineering, Scientific Programming Journal*.
[11] Jase, T. W. *MS-DOS Batch Script Writing Guide*. Version 1.
[12] Joey, B. (2004). Developing for windows on linux. *Linux Journal*.
[13] Wojtczyk, M. (2008). A cross platform development workflow for C/C++ applications. *Software Engineering Advances*.
[14] Aho, A. R. S. (1986). *Compiler Principle Technique and Tools*. Addison Wesley.
[15] Sang, K. C., Brian, P., & David, B. (2010). Platform independent programs. *Proceedings of the 17th ACM Conference on Computer and Communication Security*.
[16] Brian W. K., & Dennis, M. R. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.

Pallavi Kalyanasundaram is a final year master of engineering (M.E.) student at K. K. Wagh Institute of Engineering Education and Research, Nasik. She is doing her M.E. in VLSI and embedded systems. She has completed her B.E. in electronics and telecommunication in 2012.

She is currently pursuing internship at CREST, KPIT Technologies Ltd., Pune as a part of her M.E. Project.

Sunita P. Ugale is working as an associate professor in K. K. Wagh Institute of Engineering Education and Research, Nasik, since last 19 years. She pursued her Ph. D. from S.V. National Institute of Technology, Surat.

She has published more than 35 papers in various national and international journals. She has published 2 books Titled "Fiber optic Communication by John Wiley (2012) and "Electrical Circuits and Machines" by Central Techno (2006). She bagged "Lady Engineer Award" from Institution of Engineers' (India) – Nasik in 2008. She has worked as a BOS member of electronics engineering for Pune University. She has received a research grant of Rs. 35 lacs from DST India.

Smitha K. P. is working as a Sr. research associate in CREST, KPIT Technologies Ltd, since last 2 years. She pursued her MS from Manipal University in embedded systems and design.

Her field of interest is parallel computing. She has published papers in PDPTA'14 and SAE World Congress 2015.

Priti Ranadive is working as a principal scientist with Center for Research in Engineering Sciences and Technology (CREST), KPIT Technologies Ltd., India.

Her areas of interest include multicores, parallel programming, OS and RTOS, nature inspired innovations and TRIZ techniques. She has published more than 15 papers and is a co-inventor of two patents granted by the USPTO.