

The Windows NT™ Operating System on a PowerPC™ Microprocessor

Matt Holle
RISC Division Motorola, Inc.
Austin, TX

Abstract

This paper describes some of the major obstacles which were encountered and overcome during the port of Windows NT operating system to a PowerPC processor. The team charged with doing the port inside Motorola is quite small relative to the scope of the project. Fortunately, the modularity and portability of the operating system structure allows for a small and specialized design team to be quite effective.

Introduction

Too often in today's desktop computer marketplace, technically-correct solutions do not make it to the end user because of extraneous market influences. In this environment, it can be difficult for a single-company, of any size, to successfully take a product from conception to productization. More and more companies are combining forces to achieve success in the desktop marketplace. One of the largest examples of this is the Apple/IBM/Motorola alignment. There are few examples of single companies which are successfully providing microprocessors, systems, and operating systems. Realizing the size and complexity of the current desktop market, it was decided that the Windows NT operating system on a PowerPC processor may provide a

substantial software base in a relatively short amount of time, thus jump-starting the market for PowerPC processors.

A large software base can be vital to the success of a microprocessor in today's market. In early 1993, Motorola announced general sampling of the first implementation of the PowerPC family: the PowerPC 601™ processor. The Windows NT operating system may potentially be a dominant computing environment in the desktop computer market in the 90's. Providing the Windows NT operating system on a PowerPC processor could provide two important results: an instant software base for the PowerPC family of microprocessors, and a new performance standard for desktop users.

Windows NT Operating System Structure

The Windows NT operating system is a preemptive multitasking operating system. In other words, the operating system can be working on many tasks simultaneously. This is achieved by placing each task in a queue. Each entry in the queue is given a small amount of processor time and then placed at the end of the queue. The queue is processed so quickly and in such a way that

Windows NT is a registered trademark of Microsoft Corp.
PowerPC Architecture is a registered trademark of IBM
PowerPC is a registered trademark of IBM
PowerPC 601 is a registered trademark of IBM

each task thinks it has the entire computer all to itself.

The structure of the Microsoft Windows NT operating system can be broken into two main areas, with two additional areas outside operating system proper. For this reason, the port was approached using four teams: kernel (including the boot loader, virtual memory management, and the exception model), subsystem (including user-mode code and emulation packages), firmware, and the hardware abstraction layer (HAL) which resides between the actual hardware and the operating system. Additional teams were also formed to handle device drivers, test, system administration, and generating the software developers kit (SDK) specific for PowerPC processors. Figure 1 depicts the modularity of the NT operating system structure. Notice that none of the actual operating system (kernel and subsystems) touches the hardware. Only the HAL and firmware are aware of the hardware specifics. One exception to this case is the virtual memory management code which is located in the kernel.

Note the subsystem blocks at the top of figure 1. All user-mode threads execute inside a subsystem block. Examples of subsystems might be the Win16 emulation, Win32 application programming interface routines, DOS emulation, ... Having the subsystems located in separate functional blocks allows subsystems to be added or deleted with little or no effect on other subsystems or the kernel. This type of abstraction allows an open end for future expansion or modification as well as easing port efforts of user-mode code.

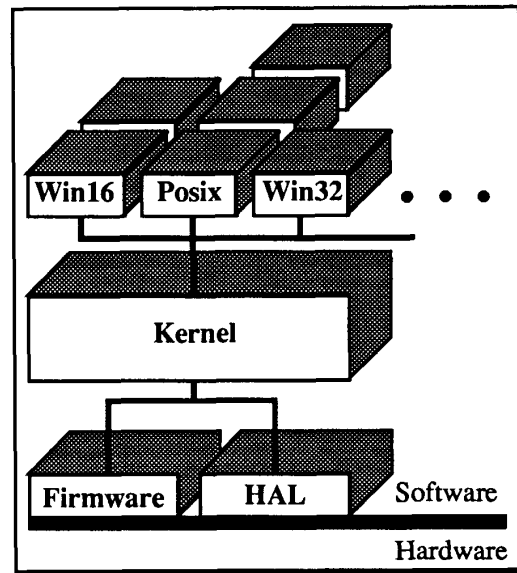


Figure 1: Windows NT Operating System Block Diagram

This paper focuses on the two teams working on the operating system proper, namely, the kernel team and the subsystem team. The list of people which contributed to these efforts is too long to list. Suffice to say that these hurdles were overcome with a cooperative team effort.

Kernel Development

The kernel is the most critical and most difficult area of a Windows NT operating system port. Two specific areas of difficulty in the kernel are defining and implementing a user-mode programming model and defining and implementing the memory management unit (MMU) functionality.

As previously mentioned, the Windows NT operating system can have many tasks running simultaneously. In this paper, the term "programming model" refers to all features of a microprocessor which are visible to a process (a single task running

under the operating system). Under the Windows NT operating system on PowerPC microprocessors, there are two kinds of processes, user-mode and kernel-mode. Kernel-mode processes have access to all resources on the microprocessor. In contrast, user-mode processes only have access to certain resources of the processor. Curtailing the resources available to user-mode processes is necessary for security sake. However, great care was also taken to ensure that restrictions were not overly stringent. User-mode debuggers must have access to enough information to debug any user-mode process. An example of this is the inclusion of a supervisor-level read-only register in the user-mode programming model which provides the address of the last instruction which caused an exception. Under typical user-mode operation, there would be no need to access this register, however, this information is vital for a debugger to implement single-stepping and breakpoint functionality.

The definition of a user-mode programming model falls within the kernel's responsibility because it is the kernel that performs all context switches for kernel processes as well as user-mode processes.

Once the definition of a user-mode programming model has been decided upon, all of the relevant kernel libraries must be modified to correctly handle the new data structure. Much of the kernel-mode code has to do with managing a queue of processes waiting for their slice of the time-share mechanism. To implement the proper processor-specific PowerPC code, one must also understand how processes are managed by the kernel. This can involve a significant learning curve.

The second area of difficulty is the definition of the memory management unit (MMU) functionality and structure. As previously mentioned, each process thinks it has the entire computer to itself. One aspect

of this is that each process thinks that it has a flat 4Gbyte address space (the Microsoft Windows NT operating system is a 32-bit operating system, thus 4Gbyte is currently the maximum address space). The MMU is responsible for mapping logical addresses (addresses which are generated by the program) to a physical address (an address which is actually present in the computer). Although each program thinks it has a 4Gbyte address space, running this operating system does not require a computer maker to actually install 4Gbyte of DRAM. The MMU knows how much actual DRAM is in the system and simply remaps each address generated by a program into different areas of actual DRAM space. If an overlap occurs, some DRAM may be temporarily written to disk.

Recall that most of the hardware specific code resides in the firmware and HAL. The MMU is an exception to that rule. Although the MMU is processor (and sometimes architecture) specific, it is kept in the kernel which may provide performance benefit on some processors. The reason for the possible performance benefit is that the MMU can often cause numerous exceptions, which can be costly to handle. Having fewer layers of abstraction (and function calls) may speed exception handling. This feature will probably not lend itself to much additional performance on PowerPC processors since flow-control is handled so efficiently on many of the current implementations.

In November 1993, kernel-mode operation of the Windows NT operating system was achieved on a PowerPC processor. This event was a significant milestone in the NT port since it proved that both the process management algorithm (user-mode processes were also able to run) and the MMU data structures were implemented correctly.

Subsystem Development

The Microsoft Windows NT operating system was developed with portability in mind. The designers allowed most subsystems to run in user-mode. A subsystem is a block of code that is grouped together by functionality. In addition, subsystems are usually kept distinct and mutually exclusive wherever possible. This promotes abstraction and keeps inter-subsystem dependencies at a minimum. Keeping the inter-subsystem dependencies at a minimum also allows for ease of integration of future subsystems. An exception to the dependency rule is the Win32 subsystem. It is from this subsystem that all user-mode processes begin (e.g. the File Manager). Other processes may be spawned off from within the Win32 subsystem into other subsystems. For example, the user may be running the File Manager and decide to fire up a DOS window. The user's request to start a DOS session would be sent to the kernel where the process manager would take control. Soon, the user would see the DOS window appear. Although the DOS window would be executing under the Win16 subsystem, the user is aware only that another process was launched.

Since it can be assumed that the Win32 subsystem is always running if there are any user-mode processes running, other subsystems can make references to functionality in the Win32 subsystem. For example, the Win16 subsystem is free to use any of the Windows NT operating system supported GDI (graphical device interface) functionality even though that functionality is not located inside the Win16 subsystem (it is located in the Win32 subsystem). In fact much of the Windows NT operating system API (application programming interface) functions are located in the Win32 subsystem. This is the only case where sharing of functionality between subsystems is promoted.

When Motorola began porting the Microsoft Windows NT operating system subsystems, we soon realized that one of the largest obstacles would be the overwhelming size of the code. The actual organization of the subsystem structures is quite straight forward and logical. A very effective means of porting a majority of this code was to simply look for processor-specific areas in the code. Recall that the subsystems run in user-mode memory, therefore, none of the code should have any knowledge of the underlying hardware. It turns out that most of these processor specific areas in the subsystems were merely request for the compiler to properly align the data structures (a requirement on some RISC microprocessors) and architecture-specifics in the build scripts. Simply searching for and modifying these processor-specific areas was quite effective because the majority of possible bugs had already been eliminated since the algorithms had been proven to work on other RISC processors.

While porting the subsystems, Motorola had to be concerned with easing the efforts of ISVs (independent software vendors) who may be porting their applications to a PowerPC processor running this operating system. Since the vast majority of applications running under the Windows NT operating system run in user-mode memory space, a port to a different platform should simply be a recompile. However, many RISC microprocessors require strict data alignment. Strict data alignment means that data (or instructions) must be aligned in memory with respect to size. For example, if an application tried to access a WORD-sized (4 bytes) piece of data, the address of that data must be evenly divisible by 4. Likewise, a HALF-sized (2-bytes) piece of data should have an address that is evenly divisible by 2.

A compiler can take care of the vast majority of data alignment issues. However, use of pointers is one example where a compiler can not preclude the

possibility of alignment violations. For this reason, it was almost impossible to develop a tool to examine source code, looking for possible alignment violations. It is important to note that the use of pointers does not necessarily imply that alignment violations may occur. On the contrary, code which makes proper use of pointers can be ported to a PowerPC processor running the Windows NT operating system with no changes.

If a data alignment violation does occur, the processor may take an exception. Several steps must be taken in order for the processor to handle the exception. First, program execution will be halted to allow all execution units on the processor to empty. Next, the entire programming model must be saved to memory. Program flow is then given over to the operating system in order to handle the exception (the data will be properly loaded by the operating system). Once this is done, the programming model is reloaded into the processor and execution continues inside the original program. Handling all this overhead can be quite expensive relative to the operating speed of the processor. In other words, a data alignment exception can be an expensive performance hit and should be avoided whenever possible.

There are many ways to solve the problem of frequent data alignment exceptions. The most straightforward solution might be to simply require the ISVs to rewrite their code to ensure alignment of data structures and proper use of pointers. As previously mentioned, the amount of work required may prohibit this choice. Another possibility is to simply allow these applications to run misaligned. This could result in the occurrence of a large number of misalignment exceptions which may detrimentally effect performance. A third alternative might be to implement an algorithm without the application even knowing about it. For example, at run time one could simply patch (replace) a data access instruction which caused a data

alignment exception with a series of instructions which access the data on smaller boundaries (bytes) which will not cause the exception. Future passes through this part of the application would then avoid the exception. Implementing something like this can be advantageous because it relieves the ISV from time consuming code modifications while reaping the benefits of fewer misalignment exceptions. However, a utility like this can also cause untold worries if program functionality might possibly be changed.

Despite these concerns, subsystem porting efforts (and general user-mode code) are quite straightforward when going from one processor running the Windows NT operating system to a PowerPC processor running this operating system.

Conclusion

A large software base can be vital to the success of a microprocessor. In hopes of promoting the PowerPC Architecture™ as a dominant desktop microprocessor in the 90s, Motorola decided to port the Windows NT operating system to a PowerPC platform. Implementing the Windows NT operating system on this processor will provide immediate access to a vast array of advanced software applications, data files, and programming paradigms. What's more, users of the Windows NT operating system on a PowerPC processor will also have the benefit of future advances from Microsoft as well as advances in PowerPC processors.

Soon computer users should be able to buy desktop computers running an advanced operating system in combination with an advanced microprocessor architecture. This combination should bring desktop processing to new heights.