Moving from Solaris 2 for SPARC to Solaris 2 for x86

by Michele Goodman

Are you a Solaris® for SPARC® developer who hasn't migrated your application to Solaris for x86? If you are, then this article is for you.

Solaris for x86 was released and began shipping at the end of May '93. While this release was based on Solaris 2.1 code for SPARC, it also includes installation and system administration features of Solaris 2.2.

Many developers porting to Solaris for x86 already have a product running on Solaris for SPARC. A recompilation of the application's SPARC code is usually all that is required, but there are a few important caveats. Software with hardware or kernel dependencies usually require changes to the source code. One class of dependency is when software relies on x86-specific or release-specific portions of the kernel such as: x86 device drivers (DDI/DKI), I/O bus architecture, or PC memory management techniques

Applications which use or generate hardware-dependent data files should allow for the difference in the byte-ordering between the x86 systems and other platforms. Binary compatibility of SVR4 applications on various hardware platforms can also be a migration issue. The binary compatibility specifications described below apply to Solaris 2.x.

The System V Release 4 Application Binary Interface (ABI) established a standard binary interface for application programs on systems that implement the interfaces defined in the System V Interface Definition, Issue 3. This corresponds to UNIX® System V Release 4.

The SVR4 ABI consists of two parts: a generic portion describing the interfaces that are constant across all machine implementations and a processor-specific portion describing the interface specific to a particular hardware architecture. Solaris for x86 will conform to the SVR4 ABI and its x86 processor specification supplement (iABI).

The System V Binary Compatibility Specification (BCS) is a binary interface for applications based upon System V Release 3. Its purpose is to establish a standard binary interface for application programs on systems that have implemented the older System V Interface Definition, Issue 2, which includes implementations of System V Release 3.2.

Like the SVR4 ABI, the BCS consists of a generic portion and a processor-specific portion. Solaris for x86 includes most of the BCS-1, but not those portions relating to XENIX™.

The native object file format for Solaris is ELF (Executable and Linking Format). In the same way that the SPARC port of the Solaris software allows dynamically linked SunOS 4.x executables using a.out object file formats to execute in a binary compatibility mode, Solaris for x86 allows most INTERACTIVE™ UNIX object files using the COFF (Common Object File Format) object file format to run in a binary compatibility mode.

The method for providing binary compatibility for COFF object files is similar to the library mapping scheme used for a.out object files. Some of the calls are provided in the kernel, some in shared libraries, and possibly some in user-level commands.

All programs consist of data structures and algorithms. As long as a program creates and consumes all of its data structures within the context of the same processor architecture, there is no problem. However, there are certain differences that exist between the SPARC and x86 processors. These differences become apparent when attempting to exchange data between the two.

Most significantly, there are differences between the ordering of the individual bytes within a data structure. Within a given 32-bit word, individual bytes are allocated from the "high" byte towards the "low" byte on the SPARC processor; but on an x86 processor, individual bytes are allocated from the "low" byte towards the "high" byte, as shown in Figure 1.

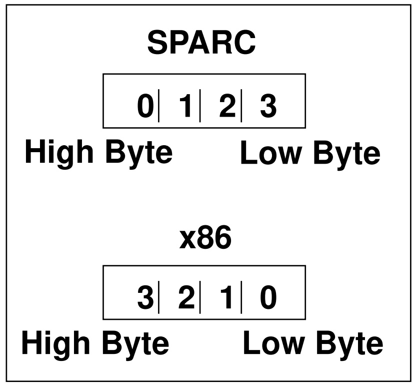

Figure 1. Byte ordering differences between SPARC and x86

Hence, the SPARC architecture is called a "Big Endian" machine, and the x86 architecture is called a "Little Endian" machine. This is the primary incompatibility in data structures between the SPARC and x86 architectures; it is at the heart of the porting issue.

Byte Alignment

In addition, there are also differences between the byte alignment of double and long double floating-point variables. The SPARC processor enforces 8-byte alignment on double float variables and utilizes a quad-word and 128-bit format for its double float variable, which are aligned upon quad-word (16 byte) boundaries. In contrast, the x86 processor enforces only 4-byte alignment upon its double float variables and utilizes a 96-bit IEEE extended precision arithmetic, which is aligned upon 12-byte boundaries.

What are the implications of these differences? First, the amount of data allocated for a structure is affected by how much space is allocated for its members. Second, unions must assume the alignment of their most strictly aligned member. The size of any object must always be a multiple of the object's alignment. Arrays of these elements will use the same alignment for all its variables. To illustrate, let's examine how identical data structures from a programmer's perspective get translated into the two architectures.

Figure 2 shows how related data types double and long double do not agree.

The critical issue lies in the size and alignment of the two data types. As mentioned earlier, on the SPARC processor, the value of a double-precision data type occupies one doubleword, and the value of a quad-precision data type occupies two doublewords. The addresses of these values are in multiples of eight (a doubleword boundary). On the x86 processor, there is no requirement for doubleword alignment for a double-precision value. The double-precision values occupy one doubleword, and their addresses occur in multiples of four.

In the byte ordering differences (Figure 1), you learned how bits are allocated. It shows that in the simplest case, there is no problem with porting between the SPARC processor and the x86 processor because the structures should align themselves the same on both architectures.

	Size of (Bytes)		Alignment (Bytes)		
Type	**SPARC**	**x86**	**SPARC**	**x86**	
Double Float	8	-	8	-	Double Precision
	8			4	Double Precision (IEEE)
Long Double Float	16	-	8	-	Quad Precision
		12		4	Extended Precision (IEEE)

Figure 2. Processor alignment differences between SPARC and x86