

ENTERPRISE NETWORKS & SERVERS

A chronicle of information for networks and servers in the enterprise

CISCO • 3COM • Juniper • Nortel • Foundry • F-5 Networks • Lucent

SUN • HP • IBM • UNISYS • DELL • SGI

Search

The magazine

This month's issue

Previous issues

More articles

Industry news

OpinionWire

TECHwatch

ARS Analyst Outlook

Resources

Book chapters

from Cisco Press

Contact us

October 2006 issue

FEATURES



ENTERPRISE
NETWORKS

Register
stay co

Email Ad

Submit!

Extend your application's reach from 32-bit to 64-bit environments Part 3: Porting example

BY CHANDRA SHEKAR

In Part 1 of this series of articles on 32- to 64-bit porting, we provided some background on the key porting challenges. In Part 2, we offered some porting guidelines. In this part, we'll look at an example of an involved porting project we completed.

Engineers at S7 ported Bristol Technology Wind/U - a C/C++ application containing more than 1.5 million lines of code - to 64-bit for HP-UX and Linux running on Intel Itanium 2 microarchitecture. Wind/U allows companies to quickly and easily port their C/C++ Windows-targeted applications to UNIX and Linux operating systems running X-Windows System based on Motif. Wind/U implements Microsoft Windows Application Programming Interfaces (APIs) and Microsoft Foundation Classes (MFCs) libraries, making calls to the X interface and Motif. The project took two engineers about three months to complete.

We designed the 32-bit version of Wind/U when Motif 1.2.x was the standard for X-Windows Systems. Unfortunately, this version was not part of the Open Group's code offerings and was not publicly available when we started our porting project. However, 64-bit Motif 2.x was available. We first had migrate Wind/U to Motif 2.x under 32-bit systems, then we could port to 64-bit environments.

Wind/U was originally written with a 32-bit environment in mind. With 64-bit architectures, pointers and longs are 64-bits in length. As we mentioned in earlier articles, this change is the source of many coding issues when porting to 64-bit. The majority of our code changes revolved around this very challenge. Throughout this article, we'll describe the issues we ran into and how we resolved them. We'll also provide generic codes samples to help clarify our code changes. You can also find some sample code listings (some compileable) at the end of the article that help illustrate the issues we ran into.

Pointers

Because pointers are 64-bits in the Intel Itanium 2 microarchitecture, we had to check and clean the code wherever a pointer was concerned. Wherever a pointer was cast as a 32-bit type, it was

truncated. The calling function accessing memory with this truncated value resulted in a system crash.

Pointer math. Wherever arithmetic code first typecast the pointer to an unsigned int or int and then returned the computed value as a truncated pointer value, any function accessing the truncated pointer value would crash the system.

For example, the following calculation

```
current_ptr = (XtPointer)((unsigned int)bundled_data + sizeof(int));
would return a truncated pointer value. We corrected the problem
with:
current_ptr = (XtPointer)((uintptr_t)bundled_data + sizeof(int));
```

We replaced the 32-bit type with a pointer-length type, which would adjust to the correct length when compiled for 32 or 64 bits.

Arrays and pointers. Wind/U contains many arrays with memory pointers and integers. The arrays were all integer types under the 32-bit environment. To accommodate pointer length, we promoted the arrays to longs.

Structs and pointers. The code also contains several structs that can hold unsigned integers or pointers at various times. For example, we used the SetWindowLong(GWL_WNDPROC) function, which we had to change to accommodate the length of pointers by substituting it with the newer SetWindowLongPtr(WGL_WNDPROC). We also had to change many macros to accommodate the longer data types.

Mixed data types. One of the largest efforts was dealing with function call returns and parameters mixing data types. We checked and changed function declarations in header files and in the many places where the call was made throughout the 1.5 million lines of code.

Assignment mismatches

Assignment mismatches can cause either data loss or lead to failed comparisons that eventually result in execution of erroneous code paths. Tracking these problems down was time consuming, because often the crash occurred in an area of code quite distant from the cause.

For example:

```
long myLong = ~0x3;
unsigned int myInt = myLong;
if(myInt == myLong) {
/* this will never execute when compiled in 64-bit */
}
```

To fix this, we either promoted the int to a long, or downcast the long to an unsigned int during the comparison. For example:

```
if (myInt == (unsigned int)myLong) {
/* this will execute when compiled in 64-bit or 32-bit */
```

```
}
```

Undeclared functions and system calls

The C grammar implies that undeclared functions return as ints. This is acceptable in the 32-bit environment. In the 64-bit world, it can lead to problems when the return is 64-bits long, like a pointer. We had to make sure each function was declared or its respective header file included before the call.

We also had to make sure that any return from a system call, such as lseek() or tell() could accept a 64-bit value.

Data issues

Alignment. In several places, Wind/U converts a data structure to a bytestream (and vice versa). We had to adjust the logic to handle this, since the word boundaries differ between 32- and 64-bit environments. The compileable alignment sample at the end of the article illustrates the differences.

Bit masking and byteswapping. We use macros for bit masking and shifting. They all had to be adjusted for 64-bit data boundaries. For example, we use the following simple macro in the 32-bit code, which causes problems for 64-bit environments.

```
#define SRCPAINT 0xFFFF0002L
```

To correct this macro for Itanium 2-based systems, we cast the literal as an unsigned int and removed the L:

```
#define SRCPAINT (unsigned int) 0xFFFF0002
```

Big-endian and little-endian issues arise when data must cross platforms between x86-based PCs and Itanium 2-based platforms. This requires byteswapping. We created macros for this function, which had to adapt when the code was compiled for different target architectures. The macros are listed in the Samples section.

Formats

We had to examine any API that obtained or displayed information using formats. Trace and log messages were typical issues we had to resolve.

For example:

```
#if !defined(_S7_64BIT)
_TRACE(_T("%s: hwnd=0x%04X, msg = %hs (0x%04X, 0x%08IX)
"),
IpszPrefix, (UINT)pMsg->hwnd, IpszMsgName,
pMsg->wParam, pMsg->IParam);
#else
_TRACE(_T("%s: hwnd=0x%08X, msg = %hs (0x%08X, 0x%08IX)
"),
IpszPrefix, (unsigned long)pMsg->hwnd, IpszMsgName,
pMsg->wParam, pMsg->IParam);
#endif
```

COM objects and virtual tables

Microsoft Windows Shell preceded COM technology but still Shell is a very important component of the MS Windows operating system. Shell is mostly written in C whereas COM, by definition, is polymorphic and the COM framework is built on the concept of virtual table. Without getting into too much depth, there was a need for the C code to call, a given C++ object's, virtual table functions. Please see the attached sample for an example of how this can be done.

When implementing MS Shell for non-windows platforms, Wind/U engineers had to figure out the exact location of the vtable within a C++ object and implement it in the form of C macros. This implementation is quite involved because every C++ compiler lays out the C++ object in a different way. There is no set standard. This was one of the main reasons for employing C macros because different Unix/Linux platforms had different compilers. This piece of logic too needed fixing when migrating from 32-bit to 64-bit platform.

Co-existence of 32- and 64-bit applications

We had to allow older 32-bit applications to easily co-exist with newer 64-bit programs on the same server. Wind/U implements a Windows registry, which stores user-specific information. All COM-based applications depend on the data in the registry, but the structure had to accommodate 64-bit values when new entries were made. Our solution was to create duplicate databases, a 32-bit version and a 64-bit version. Then we created logic to accommodate new registrations, while being user-transparent and allowing migration from the 32-bit to 64-bit databases. Our logic simply looked first in the 64-bit version of the registry for the desired data. If it wasn't found, it searched the 32-bit version, and, when found, copied the data to the 64-bit version and returned the registry value.

Summary

The major issues faced when porting 32-bit code to 64-bit environments revolve around accommodating 64-bit data, like pointers. Unfortunately, the issue appears in many places and forms throughout your code. We had to resolve many parts of the Wind/U code to cleanly port to the new 64-bit Intel Itanium 2 microarchitecture and HP-UX 11i. In addition, we had to adapt how we handled Microsoft Windows APIs and Windows registry values, since Wind/U implements Windows-targeted code under 64-bit, non-Windows operating systems.

Chandra Shekar is co-founder S7 Software Solutions, a member of the Itanium Solutions Alliance. Shekar may be contacted via the web at s7solutions.com.

Samples

The following samples help illustrate some of the issues discussed in this article. You can recreate, compile, and run compileable samples on an Intel Itanium 2 processor-based system. Other samples are

simply code listings from Wind/U we believe are valuable to the reader.

Compileable samples

These samples should be compiled on an Intel Itanium 2 processor-based system running HP-UX 11.23, using HP C++ compiler 06.05 (other compilers may also work). Compile the samples as follows:

For 32-bit: `cc -Aa +DD32 +e test.c`

For 64-bit: `cc -Aa +DD64 +e test.c`

The output is an executable named `a.out`, which illustrates the problems on 64-bit machines. To correct these problems, recompile the sample file using:

`cc -D_S7_64BIT -Aa +DD64 +e test.c`

Alignment sample. This sample illustrates how data alignment must be adjusted to word boundaries between 32-bit and 64-bit environments.

```
#include <stdio.h>
```

```
typedef struct _myStruct {
```

```
int i;
```

```
long l;
```

```
} myStruct;
```

```
int main() {
```

```
myStruct m = {10, 20L};
```

```
unsigned char *ptr = (unsigned char *)&m;
```

```
printf("I = %d
```

```
", *(int *)ptr); ptr += sizeof(int);
```

```
/* Align the pointer */
```

```
#ifdef _S7_64BIT
```

```
ptr = ((uintptr_t)ptr + 7) & (~7);
```

```
#else
```

```
ptr = ((uintptr_t)ptr + 3) & (~3);
```

```
#endif
```

```
printf("L = %ld
```

```
", *(long *)ptr); ptr += sizeof(long);
```

```
}
```

Undeclared functions. This sample shows how an undeclared function in the 64-bit environment can cause problems when the return value is 64-bits instead of the implied 32-bits.

```
#include <stdio.h>
```

```
#ifdef _S7_64BIT
```

```
extern int *AllocInt();
```

```
#endif
```

```
int main() {

int *i = AllocInt();

    • i = 100;

}
```

Sample Listing

Byteswapping. These macros addressed big endian and little endian issues.

```
#ifndef _S7_64BIT /* 32bit */
#define BYTESWAP(x) (x = (((unsigned int)((x) & 0xff) << (unsigned
int)24)
| ((unsigned int)((x) & 0xff00) << (unsigned int)8)
| ((unsigned int)((x) & 0xff0000) >> (unsigned int)8)
| ((unsigned int)((x) & 0xff000000) >> (unsigned int)24)))
#define BYTESWAP_VALUE(x) (((unsigned int)((x) & 0xff) <<
(unsigned int)24)
| ((unsigned int)((x) & 0xff00) << (unsigned int)8)
| ((unsigned int)((x) & 0xff0000) >> (unsigned int)8)
| ((unsigned int)((x) & 0xff000000) >> (unsigned int)24))
#else /* 64bit */
#define BYTESWAP(x) (x = (((unsigned long)((x) & 0xff) <<
(unsigned int)56)
| ((unsigned long)((x) & 0xff00) << (unsigned int)40)
| ((unsigned long)((x) & 0xff0000) << (unsigned int)24)
| ((unsigned long)((x) & 0xff000000) << (unsigned int)8)
| ((unsigned long)((x) & 0xff00000000) >> (unsigned int)8)
| ((unsigned long)((x) & 0xff0000000000) >> (unsigned int)24)
| ((unsigned long)((x) & 0xff000000000000) >> (unsigned int)40)
| ((unsigned long)((x) & 0xff00000000000000) >> (unsigned int)
56)))

#define BYTESWAP_VALUE (x) (((unsigned long)((x) & 0xff) <<
(unsigned int)56)
| ((unsigned long)((x) & 0xff00) << (unsigned int)40)
| ((unsigned long)((x) & 0xff0000) << (unsigned int)24)
| ((unsigned long)((x) & 0xff000000) << (unsigned int)8)
| ((unsigned long)((x) & 0xff00000000) >> (unsigned int)8)
| ((unsigned long)((x) & 0xff0000000000) >> (unsigned int)24)
| ((unsigned long)((x) & 0xff000000000000) >> (unsigned int)40)
| ((unsigned long)((x) & 0xff00000000000000) >> (unsigned int)
56))
#endif
```

This article appears in the [October 2006 issue](#) of Enterprise Networks & Servers.



[Order reprints of this article](#)

Other articles in this section

- ▶ [Looking at more on IP multicasting](#)
 - ▶ [When size does matter](#)
 - ▶ [Energy-efficient performance platforms for the enterprise](#)
 - ▶ [Reducing complexity in the data center](#)
 - ▶ [Partnering for VoIP](#)
 - ▶ [Winning the war against spyware](#)
 - ▶ [The IPAM challenge: "No known address" can mean network failure](#)
 - ▶ [Reducing complexity in the data center](#)
 - ▶ [The Rise of Rootkit-Based Malware: Why anti-spyware and anti-virus software is no longer enough](#)
 - ▶ [INETD - Internet daemon, the master server](#)
 - ▶ [Fragile trust in virtual teams threatens business performance; Research identifies new rules for communication](#)
- ▶ [**VIEW ALL ARTICLES IN THIS ISSUE**](#)

Copyright ©2003-2007 by [Publications & Communications Inc. \(PCI\)](#)
All rights reserved. Reproduction without written consent is prohibited.

[Home](#) • [Contact us](#) • [Subscriptions](#)