10 Biggest Issues for Developers Migrating 32-bit Applications to 64-bits

January 17, 2011 by John Mueller

Your company is ready to upgrade its custom applications from 32-bit to 64-bit. (Finally.) Here’s 10 tips to help you make the transition as painless as possible.

64-bit hardware has been around for a long time, but 64-bit operating systems didn’t catch on as quickly as the hardware vendors expected. There are a lot of reasons for the lag in adoption, and plenty of good reasons for applications to take advantage of the new hardware – but the bottom line is that from a usage perspective, 64-bit operating systems are only now becoming truly useful. Users now see benefits to 64-bit operating systems, so developers like you soon will migrate 32-bit software to a 64-bit environment.

Which means the project is about to be dumped on your lap. The following list describes the issues that confront developers most often when making the move.

1. 64-bit Libraries

Although it may seem like a no brainer, many application upgrades fail due to a lack of 64-bit libraries. A 64-bit application works best with 64-bit support libraries. If you use 32-bit libraries, then you must marshal data between the two environments. That slows down application speed considerably and introduces potential errors. Marshaling pointers can become especially troublesome when the pointer is for something like a data structure, which may not convert well between 64-bit and 32-bit environments.

Most problematic libraries contain custom code or come from third party vendors who add value to platforms such as Java and .NET. However, you also need to exercise care with main platforms. For example, the 64-bit version of the .NET Framework generally resides in the WindowsMicrosoft.NETFramework64 folder, rather than the WindowsMicrosoft.NETFramework folder—a difference that some developers will miss. Open source libraries may require that a recompile to provide 64-bit functionality; try to find a 64-bit download first so that you don’t have to perform the required optimization.

2. Third Party Product Support

Some applications rely on applications created by other vendors. These other applications may not work well with 64-bit applications, or the 64-bit version of the third party product might have compatibility issues. For example, consider an Excel application that accesses eBay’s Web service for the purpose of automating sales. The 64-bit version of Excel isn’t completely compatible with the 32-bit version of Excel, so you might encounter a host of unexpected problems with your application upgrade. It’s easy to miss a problem such as using a 32-bit number to hold a 64-bit handle, truncating it, and causing a crash because the handle isn’t valid.

3. Data Access

All kinds of problems can occur when moving code from a 32-bit data environment to a 64-bit data environment. One common problem is that an 64-bit application that writes 32-bit values as 64-bit values. A 32-bit application accessing the same database will access the same data in 32-bits, with resulting errors. In some cases, the data is completely mangled before anyone realizes that there is a data marshaling problem. When you perform an update, make sure that 32-bit data values are still written as 32-bit values.

Sometimes, the data marshaling requirements can be quite subtle. For example, code from a managed environment such as Java or .NET can call into the operating system. To the managed code, the value still appears to be 32-bits, but the 64-bit operating system call treats it as 64-bit data, introducing a subtle, nearly impossible to locate error. Someone in your team has to trace absolutely every data transaction to ensure that what you think you’re writing as data is what you’re actually writing. Of course, you always test your update on test data, not on production data.

Data issues include both data storage on disk, as well as in memory. Many developers learn the hard way that data structures that work fine in a 32-bit environment no longer work in a 64-bit environment. This particular problem can have many causes, but the main issue is the way the 64-bit environment packs the data into the structure. The data structure uses a different alignment in 64-bits, so that even if all of the data elements are the same size, they aren’t in the same location in memory. Of course, this problem is exacerbated by problems such as pointer size; a 64-bit pointer is twice the size of a 32-bit pointer.

4. 64-bit Programming Rules and Skills

Humans make code updates. It’s something that somehow escapes the notice of planners at some organizations, but it’s a fact that you need to consider. If your coding team lacks 64-bit programming experience, it presents a barrier to performing an application upgrade. For example, team members should be aware of the rules for converting data from 32-bit to 64-bit form, and vice versa. (What’s worse is that these rules are generally unwritten, platform and application specific, and often rely on past experience.)

If you have any hiring responsibility, ask job candidates about their experience in this area. If you’re “just a programmer” and can only deal with the people you’re dealt, pay even more attention because it’s knowledge you need to acquire (and it won’t be bad for your career if the 64-bit expert turns out to be you). If team members lack this sort of information, the resulting 64-bit application will likely contain errors that the team won’t even find. More importantly, the lack of 64-bit programming skills will cost time and money—something in short supply for most organizations today.

5. Operating System Feature Access

It may seem as if a 64-bit operating system would make it easy to access operating system features, but this isn’t always the case. For example, Windows provides an extremely odd method of allowing access to the registry. On the 64-bit version of the operating system, 32-bit applications can suddenly find their keys moved and 64-bit applications may find it difficult to locate data generated by 32-bit counterparts. The solution to this problem (and those like it) is to detect whether you’re working with a 64-bit version of the operating system, and then provide code that looks for the required 32-bit data in the alternate location.

6. Prepare to Tell the Boss: Check Hardware Capacity

A 64-bit application uses data elements that are twice as wide as a 32-bit application. The memory requirements don’t suddenly double, but the new 64-bit application will have bigger memory requirements. Suddenly, the 32-bit application that used to fit just fine in your user’s memory won’t work as a 64-bit application, and the system will run slower – which is exactly opposite of what they expect. As part of the application update, the organization should perform a hardware survey to ensure that the hardware has enough capacity to support the 64-bit environment.

7. Trick Code

Some developers rely on trick code to achieve certain goals, which might include weird hacks to work around an otherwise unsolveable defect. For example, I might add 1 to a 32-bit register until such time as the register overflows and returns to 0 (with a requisite change to the overflow flag). When making the move to 64-bits, such code can break, causing the application to malfunction. It doesn’t matter which language you use; all of us developers have created trick code of this sort. It’s a shortcut to writing longer code that doesn’t execute as efficiently (but often executes more reliably). Before making the transition to 64-bit code, perform a code review to locate potential problems of this sort. The more clever it is, the more carefully you should examine it. The trick code will have to be rewritten to execute more reliably in a 64-bit environment.

Of course, you might suggest that the added reliability causes speed problems. Users see speed; they don’t see reliability. Making the code more reliable by rewriting it to use best practice techniques also makes it more secure. So, you’re trading a bit of speed for enhanced reliability and security by rewriting the code. Given that speed is cheap today and that nefarious individuals are intent on invading your network, the rewrite is at most a tiny inconvenience for a huge payoff.

8. Math

It seems as though mathematical operations should be safe when converting from 32-bits to 64-bits. After all, 1 + 1 still equals 2 whether you use 32-bit or 64-bit numbers. The problem occurs in a number of ways. One potential problem is the issue of sign extension. A value is converted to a signed or unsigned integer, when you actually meant to use the other type of value—the equation 1 + 1 might suddenly become 1 – 1 without your knowing it (this is an oversimplification, but it demonstrates the principle). To avoid this potential problem, your test harness should consider a number of math tests for the converted application that test limits of both positive and negative numbers to ensure the application produces accurate results.

9. Supporting Hardware

Some hardware simply won’t support 64-bit access, and that’s especially so for older hardware. This concern may not affect most business applications, but it definitely affects any scientific application that relies on any sort of external sensor. It can also affect a variety of industrial and specialty applications. For example, consider a security system; you need to know that the cameras will interact properly with a 64-bit application before you perform the upgrade and find that they won’t.

10. Drivers

You might think that driver issues are gone once a 64-bit operating system can access all of the hardware on a system. However, the operating system might rely on a trick to access a 32-bit driver. In some cases, the use of a 32-bit driver causes problems for your 64-bit application. This is especially true for video or other graphics drivers. Consider the problems of accessing a color printer or other graphics equipment if all you have is a 32-bit driver. As with libraries, make sure any drivers you want to access (including things like software sensors such as Geosense for Windows) provide 64-bit access to reduce potential errors, reliability problems, and security holes.

Now you have a definite edge over your peers in migrating 32-bit applications to a 64-bit environment. By addressing these ten basic issues, you significantly improve the chances of your upgrade project succeeding.

However, this isn’t the end of the line. Most platforms and applications require attention to specific upgrade needs. Vendors such as Microsoft are usually happy to help out. For example, check out their suggestions for migrating your managed application. Check out forums as well. Java developers would do well to look at this message thread. Linux developers might want to ask their specific migration questions on LinuxQuestions.org (for example, How to check if linux kernel is 32 bit or 64 bit). The bottom line is that you need to research the upgrade carefully before you begin creating a specification for it and long before you rewrite that first line of code.