Reducing the Risks of 64-bit Application Porting With Parasoft C++test™ and Parasoft Insure++®

The Risk of 64-bit Application Porting

Until recently, the high cost of 64-bit processors, paired with their sparse operating system and application support, largely restricted 64-bit computing to memory-intensive operations (such as large database analysis, graphics, etc.) that could not operate successfully under the limitations of the 32-bit processor. Now, 64-bit processors are widely available at a reasonable cost, supported by Linux and Windows alike, and able to run 32-bit applications as well as 64-bit ones. As a result, 64-bit computing has recently become a feasible and cost-effective option for enterprise and personal users that need optimal performance and scalability.

With more and more users migrating to 64-bit servers and workstations, it's time for application developers to consider whether their applications should be recompiled for 64-bit processing. If you decide not to port, your application will typically run on most 64-bit processors, but it won't benefit from the potential performance and scalability benefits that the processor can provide to memory-intensive applications. If you do decide to port your applications to take advantage of the additional physical and virtual memory afforded by 64-bit processors, it's important to recognize the potential problems that could occur and have a practical means to prevent them.

Understanding the Potential Problems

Adding native 64-bit support to an application can be challenging—particularly if you are not aware of the potential problems you may encounter. The essence of the problem is that the source code of most applications is built around assumptions that are valid only on 32-bit platforms; however, in most cases, these assumptions are typically undocumented, and their impact on program behavior is usually unknown.

Most variations between 32- and 64-bit platforms stem from two size differences:

- The `long` type has different sizes on 32- and 64-bit systems (32 bits for most 32-bit compilers, and 64 bits on 64-bit compilers). On most 32-bit systems, `long` and `int` are the same size; on 64 bit systems, they are not.
- Pointers have a different size on 32- and 64-bit systems (32 and 64 bits, respectively). On 32-bit systems, pointers and `int` are the same size (which is widely used); on 64-bit systems, pointers and `long` are the same size.

These two differences give rise to a number of problems, which are caused by use of incorrect casts, changed type inferencing rules in assignments and expressions, truncated assignments, etc. The following are just a subset of problems that can result from size mismatches:

- Subtle but serious memory errors in the program's own memory pool managers are caused by the (now incorrect) assumption that pointers are the same size as `int`
- `long` variable values are truncated when assigned to `int` variables (garbage data)
- `long` variable values are truncated when cast as `int` variables (garbage data)
- Memory errors are caused by casts from `long*` to `int*` and vice versa
- Integer overflow errors occur when neither operand is of size `long`, but the value exceeds the maximal integer value (garbage data)

Additionally, new memory inefficiencies may arise because of changes in the byte-alignment in structs and classes. This will be compounded by the expected memory use increase that is caused by 2x pointer sizes in 64-bit applications. Suddenly, an application may consume 25% to 50% more memory, without any obvious reason. This may translate into noticeable performance

problems. Although the addressable memory on 64-bit systems is massively larger than on 32-bit ones, the actual memory in a real computer may be the same; in this case, the larger memory footprint of the program will result in more disk swapping, substantially slowing down the application.

Identifying Non-Portable Code with Coding Standards

One effective way to prevent many of these differences from causing application problems is to enable the highest level of compiler warnings, and to use portability coding standards to identify code that is likely to cause trouble when it is ported for 64-bit systems. The compiler can catch clear portability errors, but portability coding standard analysis is required to identify the more subtle problems that result from syntactically valid code that was built on assumptions which are no longer valid on 64-bit systems. Thus, before you even attempt to port code, it's recommended that you use portability coding standards to identify code that works fine on your current platform/architecture, but that might not port well.

Checking coding standards during code inspection is possible, but typically too time-consuming to be practical and too inaccurate and inconsistent to be effective. A better alternative is to check portability coding standard compliance automatically, with a tool such as Parasoft C++test, an automated coding standard analysis and unit testing tool that offers special support for developers porting applications to 64-bit systems.

Using C++test, developers can automatically identify most or all of the portability problems that escape the compiler checks. C++test identifies portability problems by parsing source code and identifying code that does not comply with applicable coding standard rules. C++test includes coding standard rules that are applicable to most 64-bit porting projects; these rules include:

- Do not assign the result of `ints` operation to `long` without casting at least one of the `ints`
- Constant assignment to `long` should not involve `int` literals
- Do not use implicit truncation
- Incompatible cast
- Do not use `i64` or `L` suffixes directly
- Avoid implicit type conversions
- Avoid pointer arithmetic
- Do not cast pointer types to non-pointers and vice versa
- Use explicit casts in mixed precision arithmetic

These “out-of-the-box” rules can be extended, or you may create your own specific porting guidelines to add to the standard set. C++test's RuleWizard feature supports the creation of custom coding standard rules that can identify virtually any code patterns that you want to flag. This way, if you learn that certain types of constructs do not port well to a certain 64-bit environment, you can create and enforce rules that automatically search for those patterns and report their specific location.

Identifying Memory Issues with Runtime Error Detection

Memory problems are among the most serious issues that surface when applications are ported to 64-bit systems. Following coding standards can help prevent these errors; however, some manifest themselves only at runtime. To ensure that they do not complicate your port, it's best to perform runtime error detection before and after porting to catch any memory and runtime issues that coding standards cannot prevent. The best way to do this is with a runtime error-detection tool such as Parasoft Insure++.

Insure++ uses mutation testing technology to automatically detect a wide variety of memory-access problems including memory corruption, memory leaks, pointer errors, I/O errors, and more. Insure++ automatically uncovers the defects in your code-- even those defects that you were not previously aware of. Insure++ detects more errors than any other tool because its technologies achieve the deepest possible understanding of the code under test and flush out even the most elusive types of problems.

Minimal preparation is required to test code with Insure++ before you port it: your application is set to run in the current environment, you probably already have a test suite for this application, and your makefile works with the current platform/architecture. After you build with Insure++, Insure++ automatically checks for a wide variety of memory-access errors and reports the exact location where each problem occurs.

When Insure++ is done testing the application, fix all of the leaks and memory corruption problems found; these types of problems are not tied to a specific platform/architecture and will probably plague you after the port if they are not fixed now. If you already have test suites created for the original application, you should run through them with Insure++; if any problems are revealed, fix them immediately.

After you recompile the code for 64-bit operation, repeat runtime error detection because the porting process is likely to cause some new problems that could not be detected before the port (for example, new memory corruption problems or different behaviors). To find such problems, run the application through Insure++ on the new platform or architecture, then fix every bug related to porting.

Conclusion

Recent studies estimate that porting (migrating, optimizing, and certifying) a software product for new processors typically takes approximately one year. By performing portability coding standard analysis and runtime error detection, you can significantly reduce the time to market / deployment for a 64-bit application, as well as reduce the porting costs. The recent 64-bit processors from companies such as AMD and Intel have made 64-bit computing a feasible option for both enterprise and personal uses. Likewise, automated checking of portability coding standards, paired with automated identification of the memory/runtime errors that commonly complicate application porting efforts, make it practical to port applications for 64-bit computing and provide application users the increased performance and scalability that the port affords.

Learning More

To read about a comprehensive strategy for reducing risks of 64-bit application porting, see R. Newman's "Removing Memory Errors from 64-bit Platforms" (*Dr. Dobb's Journal*, October 2005).

To learn more about Parasoft C++test and Parasoft Insure++, contact Parasoft as described below, or visit http://www.parasoft.com.

About Parasoft

Parasoft is the leading provider of innovative solutions for automated software test and analysis and the establishment of software error prevention practices as an integrated part of the software development lifecycle. Parasoft's product suite enables software development and IT organizations to significantly reduce costs and delivery delays, ensure application reliability and security, and improve the quality of the software they develop and deploy through the practice of Automated Error Prevention (AEP). Parasoft has more than 10,000 clients worldwide including: Bank of America, Boeing, Cisco, Disney, Ericsson, IBM, Lehman Brothers, Lockheed, Lexis-Nexis, Sabre Holdings, SBC and Yahoo. Founded in 1987, Parasoft is a privately held company headquartered in Monrovia, CA. For more information visit: http://www.parasoft.com.

Contacting Parasoft

USA

101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

Europe

France: Tel: +33 (1) 64 89 26 00
UK: Tel: +44 (0)1923 858005
Germany: Tel: +49 89 4613323-0
Email: info-europe@parasoft.com

Asia

Tel: +886 2 6636-8090
Email: info-psa@parasoft.com

Other Locations

See http://www.parasoft.com/jsp/pr/contacts.jsp?itemId=268