Guide to Application Porting From Solaris OS to Linux

IBM Corporation

July 14, 2006

Mark Brown
STSM, IBM Linux Technology Center
Ajay Sood
Staff Software Engineer, IBM Global Services, Bangalore, India
Chakarat Skawratananond
Linux on POWER Technical Consultant, IBM
Matthew Davis
Linux on POWER Technical Consultant, IBM

Special Notices

This publication/presentation was produced in the United States. IBM might not offer the products, programs, services or features discussed herein in other countries, and the information might be subject to change without notice. Consult your local IBM business contact for information on the products, programs, services, and features available in your area. Any reference to an IBM product, program, service, or feature is not intended to state or imply that only IBM's product, program, service, or feature may be used. Any functionally equivalent product, program, service, or feature that does not infringe on IBM's intellectual property rights may be used instead.

Information in this presentation concerning non-IBM products was obtained from the suppliers of these products, published announcement material or other publicly available sources. Sources for non-IBM list prices and performance numbers are taken from publicly available information including D.H. Brown, vendor announcements, vendor WWW Home Pages, SPEC Home Page, GPC (Graphics Processing Council) Home Page and TPC (Transaction Processing Performance Council) Home Page. IBM has not tested these products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

Questions on the capabilities of non-IBM products should be addressed to suppliers of those products. IBM might have patents or pending patent applications covering subject matter in this presentation. Furnishing this presentation does not give you any license to these patents. Send license inquiries, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA. All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of a specific Statement of General Direction.

The information contained in this presentation has not been submitted to any formal IBM test and is distributed "AS IS." While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. The use of this information or the implementation of any techniques described herein is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The information contained in this document represents the current views of IBM on the issues discussed as of the date of publication. IBM cannot guarantee the accuracy of any information presented after the date of publication.

Introduction

Generally, porting Solaris OS-based applications to Linux is a simple task since both Solaris OS and Linux are based on the UNIX® system. Porting often requires only a recompile with minor changes in some compiler and linker switches. Even so, differences can arise in the areas that depend on the architecture, memory maps, threading, or some specific areas like system administration or natural language support. Generally, it is when applications depend on system-specific implementations that they then require modifications. While both Solaris OS and Linux are designed to follow standards, differences in their implementations sometimes occur. This guide highlights these differences and recommends possible solutions, if an equivalent on the Linux side is not available.

This document is based on Solaris OS Version 8 and later. For Linux, this guide focuses on the Enterprise-quality distributions from Novell/SUSE (SUSE Linux Enterprise Server 9) and Red Hat (Red Hat Enterprise Linux 4). Even so, almost all of the information in this guide will also be useful when applied to almost any other modern Linux distribution.

This guide is organized as follows:

- Planning for the migration
- Development environments of Solaris OS and Linux
- Architecture-specific differences
- System-specific differences
- Performance tuning and software packaging tools available for Linux

Porting roadmap

The following steps provide a roadmap for successful migration:

Step 1: Prepare

Do research to understand the differences between the source and target platforms. While it is true in the general sense that moving from Solaris OS to Linux is much like moving from one flavor of commercial UNIX system to another, the nature and degree of differences will shape the work effort. For example, you need to know if the endianness on the source platform is different from the endianness on the target platform. If your source platform is Solaris OS/Sparc, then endianness should be under your consideration since Sparc is big-endian, while Linux on an ia32-based machine is little-endian. IBM xSeries machines are Little-Endian, while iSeries, pSeries and zSeries machines are Big-Endian.

Determine, also, if all the required third party packages (such as databases and class libraries) are available on the target platform. For 32-bit applications, consider if it is necessary to migrate to 64-bit.

Other key factors to research include

- Development environment – what compiler and build tools are you planning to use? The set of tools typically used on Linux (GNU) is also available as a set of free binaries from Sun for Solaris OS. It might make the project easier to port your development environment to a GNU-based on first on Solaris OS, and then port with an already-working build tree.
- ISO C++ - the class libraries, namespaces and other factors are going to be different.
- Threads – If your application uses Solaris OS (as opposed to POSIX) threads, there will be extra effort involved in the move.
- Internationalization – while Linux meets ISO standards, the locales and other customization data may be different.
- Shell Scripting – Linux offers shells compatible with sh, csh, and ksh, but the files and commands used by your scripts may need to be changed.

Step 2: Configure

This step involves setting up the development environment, setting environment variables, making changes to makefiles, and so on. At the end of this stage, you should be ready to start building your application. This step may require several iterations before you're ready to move to the next step.

Since the GNU development tools (standard on Linux) are also available for free from Sun, it is possible to save some time and effort by installing them onto your Solaris OS-based development system and then set up the application development environment, make files, etc. similarly to what they would be on Linux. In this way the work of porting the build tree can be done in a familiar environment.

Step 3: Build

This step involves fixing compiler errors, linker errors, and more. This document will cover compiler and linker differences in sections below. It is not unusual for code to go through several iterations of Steps 2 and 3 before a clean build is produced.

Again, building first in a GNU-based environment on your Solaris OS machine is a good way to work out these differences while in a familiar environment. The resulting binaries are also testable on Solaris OS for preliminary debugging.

Step 4: Run-time test and debug

After the application is successfully built, test it for runtime errors. This typically involves porting your applications' test methods over to Linux as well. Experience shows that much more time is spent in this stage than any other, including the porting of the application itself. Most software vendors perform a full testing and certification effort for their ports, the same as if it were an initial release of the application.

Step 5: Performance tuning

Now the ported code is running on the target platform. Monitor performance to ensure the ported code performs as expected. If not, performance tuning must be completed. The Performance Tuning section of this document provides more information about the performance tuning tools available for Linux.

Step 6: Packaging

Will you have to distribute the resulting code to others? If so, what packaging method will you want to use? Linux provides several ways to package your application such as a tarball, self-installing shell script, or RPM. The section Software Packaging provides more information about software packaging for Linux.

RPM is the package-management system widely used on Linux. Solaris OS uses pkgadmin as its package manager. The format of the package-specification template files used by pkgadmin in Solaris OS is different from the spec file used by RPM -- translating packaging information from template file into spec files requires a substantial effort. Using a software package like the InstallShield for Multiplatforms (ISMP) could deliver common packaging software across both operating systems and reduce the porting effort. By using ISMP, application developers can use a common spec file across platforms.

Development Environment

Now let's look at some of the differences in the development environments of Solaris OS and Linux, including the following:

- Makefiles
- Compiler and linker options
- Compiler Differences in Porting
- Java Technology

GNU Make versus Solaris OS Make

Most basic features are common between various versions of make. Some of the advanced functions differ. When porting an application from Solaris OS to Linux, it is likely that there will be a need to modify the makefiles.

The help information on GNU Make (access this using the command "info make") on your Linux host, includes sections on Features and Missing, that provide a detailed list of features unique to GNU make and features not supported in GNU make. To make sure that the same makefile can be used with Linux and Solaris OS it is a good idea to follow one of the following methods:

- When making changes to makefile only use features common between both versions of make. This can be verified by testing the makefile on both platforms after the porting changes. This will make porting the makefile to additional platforms simpler.
- Switch to the GNU Make on both platforms.

Special Target Support

Special Target	***Solaris OS***	***Linux***
.DEFAULT	yes	yes
.DELETE_ON_ERROR	no	yes
.DONE	yes	no
.EXPORT_ALL_VARIABLES	no	yes
.FAILED	yes	no
.GET_POSIX	yes	no
.IGNORE	yes	yes

Special Target	*Solaris OS*	*Linux*
.INIT	yes	no
.INTERMEDIATE	no	yes
.KEEP_STATE	yes	no
.KEEP_STATE_FILE	yes	no
.MAKE_VERSION	yes	no
.NOTPARALLEL	no	yes(1)
.NO_PARALLEL	yes(1)	no
.PARALLEL	yes(2)	no
.PHONY	no	yes
.POSIX	yes	no
.PRECIOUS	yes	yes
.SCCS_GET	yes	no
.SCCS_GET_POSIX	yes	no
.SECONDARY	no	yes
.SILENT	yes	yes
.SUFFIXES	yes	yes
.WAIT	yes	no

1 These two special targets have the same effect.

2 Reserved for future use, has no effect.

Make File Search Order

File	*Solaris OS*	*Linux*
./makefile	1	1
./Makefile	2	2
./SCCS/.makefile	4	6
./SCCS/.Makefile	6	8
./s.makefile	3	5
./s.Makefile	5	7
./makefile.v	-	3
./Makefile.v	-	9

File	*Solaris OS*	*Linux*
./RCS/makefile.v	-	4
./RCS/Makefile.v	-	10

LIBPATH environment variable

The environment variable for indicating the library paths are different on Solaris OS and Linux. On Linux, it is suggested to use LIBPATH in place of the Solaris OS variable LD_LIBRARY_PATH, although the latter should work.

Command Line Option Differences

Option	*Solaris OS*	*Linux*
-b	no	yes
-dd	yes	no
-h	no	yes
-i	no	yes
-l	no	yes
-m	no	yes
-o	no	yes
-v	no	yes
-w	no	yes
-C	no	yes
-D	yes	no
-DD	yes	no
-I	no	yes
-K	yes	no
-P	yes	no
-R	no	yes
-V	yes	no
-W	no	yes

Built-in makefile variables for C++

The built-in makefile variable to specify a default C++ compiler in Solaris native Make is "CCC", while for the GNU Make it is "CXX". Similarly, the default compiler flag changes from CCFLAGS to CXXFLAGS.

Libraries dependencies

On Solaris, a target name of the form: *lib((symbol))* refers to the member of a randomized object library that defines the entry point named *symbol*. The GNU Make equivalent is *lib(file.o)*.

Empty rules

Empty rules are suffix rules specified without a command. To specify an empty rule in GNU Make, put a semicolon immediately after the dependency.

Solaris: `.c.a:`

GNU Make: `.c.a: ;`

Current targets

In Solaris makefiles, the symbol $$@ is used in a dependency list to refer to the current target.

Solaris: `$(targets):$$@.o`

In GNU Make, you have to use the following

GNU Make: `$(targets):%:%.o`

SCCS and RCS files

Suffix rules that handle System V Makefiles to support SCCS files that contain '~' are not recognized by GNU Make. In Solaris, the suffix rule .c~.o will create the file x.o from the SCCS file s.n.c. GNU Make handles SCCS and RCS files differently by applying two pattern rules for extraction from SCCS or RCS in combination with general rules of rule chaining.

Conditional Macro assignment

```
$(targets) := special_flags = -g
```

On Solaris, when any target in `$(targets)` is processed, set `$(special_flags)` equal to –g. The equivalent on Linux is

```
$(targets) : special_flags= -g
```

Pattern Replacement Macro References

Solaris allows any number of % metacharacters to appear after the equal-sign. On the other hand, Linux only allows one. Here is an example,

```
ORG=one two
NEW=$(ORG:%=integer/%.o integer/%_backup.o)
all:
      @echo ORG is $(ORG)
      @echo NEW is $(NEW)
```

On Solaris, the output from the make command is

```
ORG is one two
NEW is integer/one.o integer/one_backup.o integer/two.o
integer/two_backup.o
```

For Linux, you will get this output instead:

```
ORG is one two
NEW is integer/one.o integer/%_backup.o integer/two.o integer/%_backup.o
```

A possible workaround for Linux is

```
NEW=$(ORG:%=integer/%.o) $(ORG:%=integer/%_backup.o)
```

VPATH

If a target or a dependency file is found using VPATH, then any occurrences of the word that is the same as the target name in the subsequent rules will be substituted with the actual name of the target derived from VPATH. For example,

```
VPATH=./subdir
foo.o : foo.c
      cc –c foo.c –o foo.o
```

On Solaris, if file.c is located in ./subdir, then the command

```
cc –c ./subdir/foo.c –o foo.o
```

will be executed. However, on Linux, this command will be executed instead

```
cc –c foo.c –o foo.o
```

Command Execution

Solaris Make invokes the shell with the –e argument. This flag instructs the shell to exit immediately if any program it runs returns a non-zero status. This is not true for GNU Make. However, with the –S option in GNU Make, we can achieve the same effect.

Compiler and linker

Linux, like Solaris OS, offers a high performance compiler set in addition to the GNU Compiler Collection -- the IBM XL C/C++ compiler set. Details about its licensing are available at the XL C/C++ Advanced Edition for Linux Web site. (See http://www.ibm.com/software/awdtools/xlcpp/.) Here is a brief overview of each compiler set and their advantages:

GCC

The GNU C Compiler is far and away the most commonly-used compiler for development projects on Linux. Not only is it one of the first and premier open-source applications, shipped as a part of the development tools provided with every distribution, but it is also very portable, available on almost every hardware and OS platform. GCC also allows some code prototypes that are only understood by GCC, such as GCC specific macros. However, many of these “GCC-isms" are incorporated in the IBM XL C/C++ compilers.

XL C/C++

XL C/C++ V7.0 is the follow-on release to VisualAge V6.0 for Linux. XL C/C++ compilers offer a high performance alternative to GCC, as well as a number of additional features. Fortunately, the XL C/C++ compilers produce 32- and 64-bit GNU elf objects, which are fully compatible with the objects GCC compilers produce because the XL C/C++ compilers employ the same GNU libraries, linker, assembler, and binutils as GCC. In fact, functionality formerly exclusively offered by GCC compilers has been ported to XL C/C++ to facilitate source compatibility; some GCC macros and inline functions, for example. But aside from compatibility, XL C/C++ offers unparalleled performance.

In addition to algorithmic optimization routines, XL C/C++ provides some architecture specific support not yet available in GCC -- specifically, support for VMX vector instructions featured in the IBM eServer® BladeCenter™ JS20 product. These instructions are supported by GCC 3.3 but only XL C/C++ version 7 offers automatic vectorization of unvectorized code and optimizes it for POWER VMX instructions.

As mentioned earlier, the most commonly used compiler on Linux on x86 is GNU GCC. Following is a list of widely used compiler options for the SUN Studio C/C++ compiler and the equivalent options for the GNU GCC.

Solaris OS-to-Linux equivalent compiler options

SUN Studio	GNU GCC	Description
-#	-v	Instructs the compiler to report information on the progress of the compilation.
-###	-###	Traces the compilation without invoking anything.
-Bstatic	-static	Causes the link editor to look only for files named libx.a.
-Bdynamic	(Default)	Instructs the link editor to look for files named libx.so and then libx.a when given the lx option.
-G	-shared	Produces a shared object rather than a dynamically linked executable.
-xmemalign	-malign-natural,	Specifies the maximum assumed memory alignment and the behavior of misaligned data accesses.
-xO1, -xO2, -xO3, -xO4, -xO5	-O, -O2, -O3	Optimizes code at a choice of levels during compilation. Something about O2.
-KPIC	-fPIC	Generates position-independent code for use in shared libraries (large model).
-Kpic	-fpic	Generates position-independent code for use in shared libraries (small model).
-mt	-pthread	Compiles and links for multithreaded code.
-R dirlist	-Wl, -rpath, dirlist	Builds dynamic library search path into the executable file.
-xarch	-mcpu, -march	Specifies the target architecture instruction set.

IBM also makes a C and C++ compiler for Linux (XL C/C++), as does Intel.

Compiler Differences in Porting

Sun's Forte compiler is more forgiving of non-standard constructs than the GCC or VisualAge compilers. This results in unexpected compiler errors. Some known errors are described below, along with recommended solutions.

STD C++ headers with and without .h extension

According to the new C++ standards, standard C++ headers are used without the .h extension. This means that #include <iostream> is used instead of #include <iostream.h>. The Forte C++ Compiler allows you to mix C++ standard header files with and without the .h extension. In fact, under the Forte C++ Compiler, the standard header file with .h actually includes the header files without .h and

includes using statements. This makes compiler upgrade easier. But in Linux, neither C++ compiler does allow this mixing. The .h extension of C++ standard header files for Linux must be removed.

Namespace

Namespace is very strict with the VisualAge compilers, especially within the Standard Template Library (STL). Expressions like `using namespace std`, `using std::string`, or `using std::vector`, etc. are required in the header files to avoid compiler errors.

Containers

Some containers such as `map` and `vector` do not allow `const` type by the VisualAge compiler. Specifically, `std::map< const std::string, Foo>` will not compile with VisualAge. The `const` is unnecessary since `map` keys are immutable anyway. The `key` type is made `const` when a `(key, value)` pair is returned, but it should not be declared `const` in the `map` template.

Casting

Casting is strictly enforced by the GCC and VisualAge compilers. For example, `const void *` cannot be cast into non-const pointers.

char constant reference

Solaris OS Code	Linux Code

`#include <iostream.h>`
`class test{`
`  public:`
`// Using a constant ref. to a`
`// variable whose value can be`
`// changed`
`    void test1(char& const nEvt){`
`    cout << nEvt << endl;`
`    }`
`};`
 	
`int main()`
`{`
`  test t;`
`  char * const ptr="ABC";`
`  t.test1(*ptr);`
`  return 0;`
`}`
	`#include <iostream.h>`
	`class test{`
	`  public:`
	`// Using a constant ref. to a`
	`// variable whose value can be`
	`// changed`
	`    void test1(char const & nEvt){`
	`    cout << nEvt << endl;`
	`    }`
	`};`
	
	`int main()`
	`{`
	`  test t;`
	`  char * const ptr="ABC";`
	`  t.test1(*ptr);`
	`  return 0;`
	`}`

The `char& const` is different from `char const &`. `char& const` is a reference to a

constant value, while in `char const &`, the reference is constant but not the value of the reference.

const reference

The following code fragment would compile clean on Solaris OS, but gives error messages on Linux on iSeries and pSeries.

Solaris OS Code	Linux Code

class T {
  public:
  T(int);
};
 	
class U {
  public:
  U(T& k=T(0));
};
	class T {
	  public:
	  T(int);
	};
	
	class U {
	  public:
	  U(const T&k=T(0))
	};

Here are the errors messages that result with the VisualAge compiler:

```
"test.cpp", line 10.16: 1540-1280 (S) An rvalue of type "T"
cannot be converted to "T &".
"test.cpp", line 10.16: 1540-1290 (I) An rvalue cannot be
converted to a reference to a non-const type.
```

In order to get this code fragment to compile clean on Linux on iSeries and pSeries, `const` must be added to the parameter. The problem with making the parameter non-const is that the constructor could try to update the reference. Since the reference points to a temporary, the update would then be lost. The way to make this code fragment compile on Linux is to use `U(const T&k=T(0))`. This makes the code more C++ conformant also.

Storing returned object in a reference

The following code fragment would compile clean on Solaris OS (in the left-hand column), but gives error messages on Linux on iSeries and pSeries. In the righthead column, two solutions are shown that will compile clean on Linux on iSeries and pSeries.

Before	After

class foo {
  public:
  foo();
};
 	
foo getFoo( ) {
  foo myFoo;
  return myFoo;
}
 	
void myFunc( ) {
	class foo {
	  public:
	  foo();
	};
	
	foo getFoo( ) {
	  foo myFoo;
	  return myFoo;
	}
	
	void myFunc( ) {

Before	After

`foo& myFoo = getFoo();`
 	
`}`
	Solution 1:
	`const foo & myFoo = getFoo();`
	
	Solution 2:
	`foo tmpFoo =getFoo();`
	`foo& myFoo = tmpFoo;`
	`}`

A temporary, such as produced by the expression `getFoo()`, cannot bind to a non const reference, according to the C++ standard. Both Linux compilers adhere to that requirement.

Copy constructors, equals operator and comparison operator

VisualAge requires copy constructors, equals operator and comparison operator parameters to sometimes be const. Therefore, the general rule would be to make all the parameters const if possible to avoid errors.

Before	After

`Foo(Foo&)`
`Foo & operator = (Foo &)`
`Boolean operator == (Foo &,Foo &)`
	`Foo(const Foo&)`
	`Foo & operator = (const Foo &)`
	`Boolean operator == (const Foo &, const Foo &)`

Method naming

Method names cannot be class names.

Before	After

`class Foo;`
`class FooNext {`
`Foo *Foo;`
`};`
	`class Foo;`
	`class FooNext {`
	`Foo *fooPtr;`
	`};`

Friend usage

Using friend does not make it a class. Forward declare a class prior to making it friend.

Before	After

`class Foo{`
`  friend class FooFriend;`
`};`
	`class FooFriend;`
	`class Foo {`
	`  friend class FooFriend;`
	`};`

`Static` *friend usage*

Solaris OS' Forte C++ compiler allows the friend function to be static, but the VisualAge C++ compiler does not allow it. Do not specify the friend function as `static`.

Class Qualifiers

VisualAge does not allow usage of class qualifier :: within a class definition.

Before	After

`class FooNext {`
`  void FooNext::getFoo();`
`};`
	`class FooNext {`
	`v  oid getFoo();`
	`};`

Text after `#endif`

Neither compiler allows extra non comment text following `#endif` statements.

Before	After
`#endif extra comment`	`#endif //extra comment`

Linked Lists

The following code fragments compiles clean on Solaris OS, but give the error message below on Linux:

```
/* Type definition for a linked list and linked list pointer. */
class LinkList : public Node {
  int _listlength; /* no. of elements in linked list */
  int _itemsize; /* size of the data _item (see _NODE) */
  public :
    Node *_head; /* pointer to _head of linked list */
    Node *_tail; /* pointer to _tail of linked list */
    Node *_current; /* pointer to _current node */
}

LinkList operator++(){
  if( _current->_next )
    _current = _current->_next;
  else
    _current = NULL;
  return *this;
}

LinkList LinkList::operator=(LinkList &from_list)
{
  char *from_item,
  *to_item;
  LinkList _list(sizeof(this->_head));
  from_list++;
}
```

Error message for the "`from_list++`" line:

```
1540-0218 (S) The call does not match any parameter list for
"operator++".
1540-0215 (I) The wrong number of arguments have been specified
for
"LinkList::operator++()".
1540-1283 (I) "LinkList::operator++()" is not a viable candidate.
```

To make this work on Linux, change the declaration from

```
LinkList operator++() {
```

to

```
LinkList operator++(int ){
```

If the compiler sees `++b`, it generates a call to `B::operator++( )`. If it sees `b++` it calls `B::operator++(int)`.

C++ Style Comments

To enable C++ style comments in C source file with the VA compiler, use the flag `-qpluscmt`. This is not required for GCC.

Java technology

Java technology allows developers to deploy across multiple platforms without recompiling code, provided that the Java Runtime Environment is compatible with the Java Developer Kit (JDK) for the platform on which the software was developed. Linux Java development is supported on both RHEL and SLES by the IBM Developer Kit for Linux, Java Technology Edition. This developer kit is available in both 32- and 64-bit versions at no cost from IBM . The IBM developer kit is also packaged with both RHEL and SLES media, and updates are supported by the distributions' maintenance packages.

Architecture-Specific Differences

Now, let's look at the architecture- and system-specific differences between Solaris OS and Linux, including base data types and endianness.

Base data type and alignment

There are two different classes of data types available on a system: base data types and derived data types. *Base data types* are all data types defined by the C and C++ language specification. Table 3 compares base data types for Linux on x86 and Solaris OS on SPARC.

Comparing Linux and Solaris OS base data types

	Linux x86	**Linux IA64**	**Linux Power**	**Linux Power**	**Linux zSeries**	**Linux zSeries 64bit**	**Solaris OS Sparc 32bit**	**Solaris OS Sparc 64bit**
Base type	ILP32 (bytes)	LP64 (bytes)	ILP32 (bytes)	LP64 (bytes)	ILP32	ILP64	LP32 (bytes)	LP64 (bytes)
char	1	1	1	1	1	1	1	1
short	2	2	2	2	2	2	2	2
Int	4	4	4	4	4	4	4	4
float	4	4	4	4	4	4	4	4
long	4	8	4	8	4	8	4	8
pointer	4	8	4	8	4	8	4	8
long long	8	8	8	8	8	8	8	8
double	8	8	8	8	8	8	8	8
long double	12	16	8/16*	8/16*	8	8/16**	16	16

**The default size for long double in Linux on POWER is 64 bits. They can be increased to 128 bits if the compiler option* –qlongdouble *in XL C/C++ compiler is used.*
*** 16 bit is a future enhancement.*

When porting applications between platforms or between 32-bit and 64-bit modes, you need to take into account the differences between alignment settings available in the different environments to avoid possible degradation in performance and data corruption. Table 4 shows the alignment values in bytes for Linux on x86.

Alignment values (in bytes) for Linux on x86

Data type	**Linux IA-32 (ILP32)**	**Linux IA-64 (LP64)**	**Linux Power (ILP32)**	**Linux Power (LP64)**
Bool	1	1	1	1
Char	1	1	1	1
wchar_t	4	4	2	4
int	4	4	4	4
Short	2	2	2	2
long	4	8	4	8
long long	8	8	8	8
Float	4	4	4	4
Double	4	8	8	8
long double	4*	16	8	8

Note: *Alignment depends the compiler switches being used. These switches control the size of the "long double" type. The i386 application binary interface specifies the size to be 96 bits, so -m96bit-long-double is the default in 32-bit mode. Modern architectures (Pentium and newer) would prefer "long double" to be aligned to an 8- or 16-byte boundary. In arrays or structures conforming to the ABI, this would not be possible. So specifying a -m128bit-long-double will align "long double" to a 16-byte boundary by padding the "long double" with an additional 32-bit zero.*

System-derived data types

A derived data type is a derivative or structure of existing base types or other derived types. *System-derived data types* can have different byte sizes depending on the employed data model (32-bit or 64-bit) and the hardware platforms. Table 5 shows some of the derived data types on Linux that are different from those on Solaris OS.

Derived data types on Solaris OS and Linux

OS	gid_t	mode_t	pid_t	uid_t	wint_t
Solaris OS ILP32 l	long	unsigned long	long	long	long
Solaris OS LP64	int	unsigned int	int	int	int
Linux ILP32	unsigned int	unsigned int	int	unsigned int	unsigned int
Linux LP64	unsigned int	unsigned int	int	unsigned int	unsigned int

Endian-ness

Endianness issues are often encountered during the process of migrating applications from one type of architecture to another. Endianness is the ordering of a data element and its individual bytes as they are stored and addressed in memory. There are two types of endianness: big-endian and little-endian. Some processors (such as Sparc and Power or zSeries) store values in memory Big-Endian, from most significant to least significant value:

```
0x4A3B2C1D  is stored as 0x4A 0x3B 0x2C 0x1D
```

Little-Endian processors (such as Intel ia32) store them from least- to most-significant:

```
0x4A3B2C1D is stored as 0x1D 0x2C 0x3B 0x4A
```

Code that expects a certain address order for binary data will not be portable between these processor types. Since both SPARC and POWER architectures are big-endian, there will be no endianness issues when porting applications between these platforms. However, if you are porting Solaris OS on Sparc applications to Linux on xSeries (ia32), you may need to deal with endianness portability issues since ia32 is a little-endian architecture. Most problems arise from three types of incompatibility:

1. Non-uniform data reference
2. Data sharing across BE and LE platforms
3. Data exchange between network devices (e.g. IP and PCI)

Non-uniform data reference is chief among these issues, especially in user space application code, whereas the latter two categories are difficulties at lower code levels, e.g. device drivers. Consistent with the scope of this document, only the former is discussed in detail.

Non-uniform Data Reference

Non-uniform data reference arises from improper datatype reference with regard to endianness, usually dealing with unions or pointers. Endian-friendly code should incorporate definitions to determine if the platform is LE or BE.

It is considered good programming habit to never cast a pointer to an int and to explicitly reference datatype and byte values during conversion.

32 to 64 bit Migration

Any attempt to migrate from a 32 bit platform to 64 bit Linux should be treated as two separate ports: first from the native platform to 32 bit Linux, and second from 32 to 64 bit. Datatype mismatches are common among code incompatibilities due to endianness and 32 to 64 bit issues. Opportunities to isolate the source of these incompatibilities should be exploited at every turn.

A 64 bit application environment can radically improve the performance of memory addressing and throughput for applications that manipulate very large data structures, e.g. databases. However, underutilization of a 64 bit address space also results in performance loss.

Datatype Consistency

In a 64 bit environment, long and ptr data types are 64 bit, not 32. Failure to account for this difference from a 32 bit environment will result in compile errors when ints are improperly matched. A long is not an int in 64 bit code, and thus the two cannot be interchanged. Attempts to do so will either result in compile errors or worse yet,
data truncation at runtime. Explicit reference is key to avoiding these types of errors. Consider this code example:

Explicit reference example

```
extern long dosomething(int);

int main(int argc, char *argv[])
{
int i1, i2, i3;
long l1, l2, l3;

/* implicit truncation occurs in the next 3 statements */
i1 = l1;
i2 =i2 *l2;
i3 = dosomething(l3);

/* use explicit casting to obtain the intended narrowing */
i1 = (int) l1;
i2 = (int) i2 * l2;
i3 = (int) dosomething((int) l3);
}
```

Never case a pointer to an int in 64 bit code. This is considered by some to be poor coding practice regardless, but is forbidden in 64 bit code. Use of longs and explicit type reference are again keys to

avoiding these mistakes.

In preparing code for migration to 64 bit, the VA compilers can help identify potential incompatibilities. The -qwarn64 build flag of the VA compilers will identify potential data truncations arising from long to int conversion.

System-specific Differences

In this section, learn about the differences between Solaris OS and Linux, including system calls, signals, data types, and threading libraries.

System calls and library functions

While Solaris OS and Linux both have the same UNIX-system-based roots, they are different and some Solaris OS system calls and library functions are not available on Linux. When this happens, a wrapper-call may have to be implemented on the Linux side. Here are a few examples of incompatibilities between Solaris OS and Linux:

- **regexec() and regcomp()**: The routines `regexec()` and `regcomp()` on Solaris OS need to be replaced by `regexp()` and `regcmp()` on Linux.

- **Dirent** structure on Solaris OS is different from that on Linux.

- **File system interface routines**: Solaris OS file system routines employ `vfstab` structures and contain `vfs` in the function name, such as `getvfsent`. Linux provides equivalent interfaces, but the routines use `fstab` structures and contain `fs` in the routine name, such as `getfsent`. The `vfstab` structure on Solaris OS is defined in `/usr/include/sys/vfstab.h`. The definition of `fstab` on Linux is in `/usr/include/fstab.h`.

- **Device Information Library Functions** (`libdevinfo`): The `libdevinfo` library contains a set of interfaces for accessing device configuration data, such as major and minor numbers. The standard Linux installation does not support this.

- **CPU Affinity**: By default, a process in a multiprocessor system bounces among several CPUs. By explicitly binding a process to certain CPUs (assigning CPU affinity) may yield performance gain in some cases. System calls for CPU affinity on Solaris OS are different from those on Linux. The Linux 2.6 kernel provides `sched_setaffinity()` and `sched_getaffinity()` for CPU affinity. Please consult the Linux man pages for more information.

The following table lists the Solaris OS system calls that either use a different name, signature, or are not available on Linux:

Solaris OS system calls

Solaris OS	Linux	Notes
acl, facl	N/A	get or set a files's Access Control List (ACL)
adjtime	N/A	correct the time to allow synchronization of the system clock
audit	N/A	write a record to the audit log
auditon	N/A	manipulate auditing
auditsvc	N/A	write audit log to specified file descriptor
getacct, putacct, wracct	N/A	get, put, or write extended accounting data
getaudit, setaudit, getaudit_addr, setaudit_addr	N/A	get and set process audit information
getauid, setauid	N/A	get and set user audit identity
getdents	getdents**	read directory entries and put in a file system independent format. Dirent structure on Linux and Solaris OS are different.
getmsg, getpmsg	N/A	get next message off a stream
getpflags, setpflags	N/A	get or set process flags
getppriv, setppriv	N/A	get or set a privilege set
getustack, setustack	N/A	retrieve or change the address of per-LWP stack boundary information
issetugid	N/A	determine if current executable is running setuid or setgid
llseek	_llseek	move extended read/write file pointer
_lwp_cond_signal, _lwp_cond_broadcast	N/A	signal a condition variable
_lwp_cond_wait, _lwp_cond_timedwait, _lwp_cond_reltimedwaid	N/A	wait on a condition variable
_lwp_info	N/A	return the time-accounting information of a single LWP
_lwp_kill	N/A	send a signal to a LWP
_lwp_mutex_lock, _lwp_mutex_unlock, _lwp_mutex_trylock	N/A	mutual exclusion

Solaris OS	**Linux**	**Notes**
_lwp_self	N/A	get LWP identifier
_lwp_sema_wait, _lwp_sema_trywait, _lwp_sema_init, _lwp_sema_post	N/A	semaphore operations
_lwp_suspend, _lwp_continue	N/A	continue or suspend LWP execution
memcntl	N/A	memory management control
meminfo	N/A	provide information about memory
mount	mount*	mount a file system. The signatures of this system call on Solaris OS and Linux are different.
msgids	N/A	discover all message queue identifiers
msgrcv	msgrcv*	message receive operation. Linux uses struct msgbuf type in one of its argument.
msgsnap	N/A	message queue snapshot operation
msgsnd	msgsnd*	message send operation. Linux uses struct msgbuf type in one of its argument.
ntp_adjtime	N/A	adjust local clock parameters
ntp_gettime	N/A	get local clock values
open, openat	open*	open a file. openat() is not available in Linux.
pcsample	N/A	program execution time profile
p_online	N/A	return or change processor operational status
priocntl	N/A	process scheduler control
priocntlset	N/A	generalized process scheduler control
processor_bind	sched_setaffinity	bind LWPs to a processor
processor_info	N/A	determine type and status of a processor
pset_bind	N/A	bind LWPs to a set of processors
pset_create, pset_destroy, pset_assign	N/A	manage sets of processors
pset_info	N/A	get information about a processor set
pset_list	N/A	get list of processor sets
pset_setattr, pset_getattr	N/A	set or get processor set attributes
putmsg, putpmsg	N/A	send a message on a stream

Solaris OS	Linux	Notes
rename, renameat	rename*	change the name of a file. Linux does not have renameat().
resolvepath	N/A	resolve all symbolic links of a path name
semids	N/A	discover all semaphore identifiers
setrctl, getrctl	N/A	set or get resource control values
settaskid, gettaskid, getprojid	N/A	set or get task or project IDs
shmids	N/A	discover all shared memory identifiers
sigsend, sigsendset	N/A	send a signal to a process or a group of processes
__sparc_utrap_install	N/A	install a SPARC V9 user trap handler
fstatat	N/A	get file status
swapctl	N/A	manage swap space
uadmin	N/A	administrative control
unlink, unlinkat	unlink*	remove directory entry. Linux does not have unlinkat().
futimesat	N/A	set file access and modification times. It resolves the path relative to the fildes argument.
waitid	N/A	wait for child process to change state
yield	sched_yield	yield execution to another lightweight process

Signals

Linux supports both POSIX standard signals and POSIX real-time signals.

Note that for Linux:

- SIGABRT and SIGIOT are identical
- SIGCLD and SIGCHLD are identical
- SIGPOLL and SIGIO are identical
- Default action for SIGPWR for Linux is to terminate the process, but ignore on Solaris OS.

The following signals are supported by Solaris OS, but not by Linux:

Signals supported by Solaris OS

Name	Default action	Description
SIGEMT	core	Emulation trap
SIGWAITING	ignore	Concurrency signal used by threads library
SIGLWP	ignore	Inter-LWP signal used by threads library
SIGFREEZE	ignore	Checkpoint suspend
SIGTHAW	ignore	Checkpoint resume
SIGCANCEL	ignore	Cancellation signal used by threads library
SIGLOST	ignore	Resource lost (but supported in Linux running on Sparc)
SIGXRES	ignore	Resource control exceeded

`sigset_t` is defined differently on Solaris OS and Linux.

Solaris OS Threads and POSIX Threads

This section addresses issues that developers face when migrating multithreaded applications from Solaris OS to Linux. Solaris OS supports two threading implementations: Solaris OS threads and POSIX threads (pthreads). Linux supports the POSIX threading library (pthreads). Applications coded to the POSIX threading model on Solaris OS should not have any problems in this area, on Linux. There are some functions implemented by the Solaris OS threads API that are not implemented by the pthreads API, and vice versa. For those functions that do match, the associated arguments might not. The following features are exclusively supported by the Solaris OS threads API:

- The ability to create daemon threads. Daemon threads do not affect the process exit status. A process can exit either by calling exit(), or by having every non-daemon thread in the process call thr_exit().
- The ability to suspend or continue the execution of the thread using thr_suspend() and thr_continue(). Note that thr_suspend() suspends the target thread with no regard to the locks that the thread might be holding. If the suspending thread calls a function that requires a lock held by the suspended target thread, deadlock occurs.
- The ability to allow a thread to wait for any undetached thread in the process to terminate. This is achieved when the first argument of thr_join() is set to 0. If you set the first argument of pthread_join() to 0, the program will be terminated with a segmentation fault.

The following table compares the key Solaris OS threads functions with the pthreads functions. For other Solaris OS threads functions, please consult the "Multithreaded Porting Guide" from Sun. (See http://docs.sun.com/app/docs/doc/816-5137)

Threads and pthreads functions

Solaris OS threads API	Linux POSIX threads API	Description
thr_create()	pthread_create()	Creates a new thread of control.
thr_exit()	pthread_exit()	Terminates the execution of the calling thread.
thr_join()	pthread_join()	Suspends the calling thread until the target thread completes.
thr_kill()	pthread_kill()	Sends a signal to another thread.
thr_self()	pthread_self()	Returns the thread ID of the calling process.
thr_yield()	sched_yield()	Makes the current thread yield to another thread.
thr_getprio()	pthread_getschedparam()	Retrieves a thread's priority parameters.
thr_setprio()	pthread_setschedparam()	Modifies a thread's priority parameters.
thr_getspecific()	pthread_getspecific()	Binds a new thread-specific value to the key.
thr_setspecific()	pthread_setspecific()	Binds a new thread-specific value to the key.
thr_getconcurrency()	pthread_getconcurrency()	Gets thread concurrency level.
thr_setconcurrency()	pthread_setconcurrency()	Sets thread concurrency level.
thr_sigsetmask()	pthread_sigmask()	Changes or examines the calling thread's signal mask.
thr_keycreate()	pthread_key_create()	Creates a key that locates data specific to a thread.
N/A	pthread_key_delete()	Deletes a key that locates data specific to a thread
thr_suspend()	N/A	Suspends the execution of the specified thread.
thr_continue()	N/A	Resumes the execution of a suspended thread.
fork1()	fork()	Regular fork
forkall()	N/A	Replicate all fork

The behavior of fork() in Solaris OS 9 and earlier releases is different from the behavior of fork() in POSIX threads. In POSIX threads, fork() creates a new process, duplicating the complete address space in the child. However, it duplicates only the calling thread in the child process. Solaris OS threads API also provides the replicate all fork semantics, forkall().

This function duplicates the address space and all the threads in the child. This feature is not supported by POSIX thread standard.

There are some POSIX thread extensions implemented in Solaris OS, but not in Linux, and vice versa. The following table lists those routines:

POSIX thread extensions

Routine	Solaris OS	Linux
pthread_cond_reltimedwait_np	y	n
pthread_mutexattr_getprioceiling	y	n
pthread_mutexattr_getprotocol	y	n
pthread_mutexattr_getrobust_np	y	n
pthread_mutexattr_setprioceiling	y	n
pthread_mutexattr_setprotocol	y	n
pthread_mutexattr_setrobust_np	y	n
pthread_mutex_consistent_np	y	n
pthread_mutex_getprioceiling	y	n
pthread_mutex_reltimedlock_np	y	n
pthread_mutex_setprioceiling	y	n
pthread_rwlock_reltimedrdlock_np	y	n
pthread_rwlock_reltimedwrlock_np	y	n
pthread_setschedprio	y	n
pthread_attr_getaffinity_np	n	y
pthread_attr_setaffinity_np	n	y
pthread_cleanup_pop_restore_np	n	y
pthread_cleanup_push_defer_np	n	y
pthread_getattr_np	n	y
pthread_kill_other_threads_np	n	y
pthread_rwlockattr_getkind_np	n	y
pthread_rwlockattr_setkind_np	n	y
pthread_timedjoin_np	n	y
pthread_tryjoin_np	n	y

POSIX threading on Linux implements the one-on-one threading model (where there is a one-to-one relationship between user threads and kernel threads). It also implements the inter-process POSIX synchronization primitives. Specifically, the thread option PTHREAD_PROCESS_SHARED is supported. By default, each thread is created with the detachstate attribute set to PTHREAD_CREATE_JOINABLE, scheduling policy set to SCHED_OTHER, and no user-provided stack.

The following table shows the default values of attributes for POSIX threads on Linux:

Default values of POSIX threads attributes on Linux

Attribute	**Default Values**
scope	PTHREAD_SCOPE_SYSTEM
detachstate	PTHREAD_CREATE_JOINABLE
schedparam	0
inhiritsched	PTHREAD_EXPLICIT_SCHED
schedpolicy	SCHED_OTHER

Testing and Performance Tuning

Once the code has been ported and successfully executed on Linux, testing, performance monitoring and tuning procedures can ensure that the ported code performs well on the target platform. This section provides you with a list of tools available on Linux to help accomplish that.

GNU Development tools

The GCC compiler set, and the GNU development utilities that come with each Linux distribution, offer free and basic toolkit for debugging and profiling an application.

The basic profiling tools in Linux are the –p (profile) and –pg (profile for gprof) options in `gcc`, and the `prof` and `gprof` utilities. Compiling using –p or `-pg` causes `gcc` to insert instructions necessary to obtain profiling information into the object code. Running the prof command with the application will allow you to then obtain:

- each procedure, ordered by descending processor activity
- the percentage CPU time used by each procedure
- the execution time in seconds for all references
- the number of times each procedure was called
- the average time for a call to the procedure

Running the `gprof` command with the application will gather (among other information):

- the percentage of CPU time used by each procedure and its calling tree
- a time breakdown for each procedure and what it calls
- the number of times a procedure was called
- what procedures were called by each procedure

Since `gprof` includes the descendants of a procedure in its timings, it is more useful for procedures calling library routines.

There are other profiling tools available to the developer, prof and gprof are mentioned here as being the most commonly available. Some of these alternative tools may be specialized for a specific purpose, such as parallel programming, or for massively-multithreaded applications.

Compiler Options as Tuning Tools

Modern compilers offer many optional optimization features, `gcc` as one example offers over sixty options related to performance optimization. These compilers often "know" the machine architecture and processor better than the developer, performing as a matter of course such optimizations as dead code elimination, loop unrolling, branch optimization, and function inlining. In many cases, the programmer will find all of the major and obvious bottlenecks identified via profiling, resolved by using the optimizations provided by the compiler.

In `gcc`, the general level of optimization is controlled by the `–O` flag. At its most basic level (`-O1`), will take the most general steps to reduce code size and execution time. `–O2` will cause `gcc` to perform nearly all optimizations that do not involve a space-speed tradeoff (such as loop unrolling or function inlining). `–O3` turns on such additional optimizations as function inlining and register renaming. We recommend an interactive process of increasing optimization/profiling cycles to determine the best level for your application. This is because it is possible, even with well-written code, for a higher level of compiler aggressiveness in optimization to hurt a piece of code's performance, rather than help it. This is sometimes a matter of what the more aggressive optimizer is looking for as opposed to what the code is actually trying to do. A reading of the performance options section of the gcc manual will reveal many additional options, most of which are more suitable for use once a particular code section has been analyzed and its problems made clear.

GCC also offers a number of processor-specific performance options.

Rational Software Development Platform

The IBM Rational Software Development Platform (SDP) offers the most complete set of offerings to build, integrate, modernize, extend, and deploy software and software-based systems. It provides everything you need to automate and integrate your software development projects. See http://www.ibm.com/software/rational/ for details.

Performance Inspector

This suite of tools can be used to identify performance problems in your application (C/C++/Java) and shows how your application interacts with the Linux kernel. It consists of seven tools:

- *TProf* is a timer profiler that identifies what code is running on the CPU during a user-specified time interval. It is used to report hot-spots in applications as well as the kernel. Basically, it records which code is running at each system-clock interrupt (100 times per second per CPU). Oprofile (below) also provides this feature.

- *PTT* collects per-thread statistics, such as number of CPU cycles, number of interrupts, and number of times that the thread was dispatched.
- *AI* displays CPU utilization statistics during a user-specified interval.
- *JLM* provides statistics on locks based in the Java 2 technology.
- *JProf* is a shared library that interfaces with Java jvmpi interface.
- *POST* generates reports based on outputs from other tools.
- *A2N* is used by POST to map code execution to the application that was being traced.

At the time of this writing, Performance Inspector supports SUSE LINUX Enterprise Server 9 and Red Hat Enterprise Linux 3.0 Update 2. To download Performance Inspector and get more information, please visit the Performance Inspector home page. (See http://perfinsp.sourceforge.net/)

Post-Link Optimization for Power

This tool optimizes the executable image of a program by collecting information on the behavior of the program while the program is used for some typical workload. It then re-analyzes the program (together with the collected profile), applies global optimizations (including program restructuring), and creates a new version of the program that is optimized for that workload. The new program generated by the optimizer typically runs faster and uses less real memory than the original program.

At the time of this writing, Post-Link Optimization for Power tool is supported on the following Linux distributions: SUSE LINUX Enterprise Server 8 and above, Red Hat Enterprise Linux 3. To download and get more information, please visit the IBM alphaWorks Post-Link Optimization for Linux on POWER site. (See http://www.alphaworks.ibm.com/tech/fdprpro).

Oprofile

Oprofile provides profiles of code based on hardware-related events such as cache misses or CPU cycles. For example, Oprofile can help you determine which of the source routines causes the most cache misses. Oprofile utilizes hardware performance counters provided in many CPUs including IBM Power. Please visit the Oprofile Web site for more information. (See http://oprofile.sourceforge.net/news/).

Software packaging

Application software is delivered to end users in a unit called a package. A package is a collection of related files (binaries, libraries, documentation, and source code) and metadata. Metadata is used by the package management system to coordinate all of the pieces in the package. Solaris OS uses pkgadmin as its package manager. RPM is the package management system widely used on Linux. For further information about RPM, please visit http://www.rpm.org/ or type “man rpm” in Linux.

Note that the format of the package specification template files used by pkgadmin in Solaris OS is different from the spec file used by RPM, and translating packaging information from template file into spec files requires a substantial effort.

Summary (and Further Information)

The porting effort from Solaris OS to Linux in most cases is usually the same as the move from Solaris OS to any other “flavor” of commercial UNIX system. While there are great similarities, the differences must be taken into account. Usually, this involves just a recompile or minor changes in compiler/linker switches. A move to Linux provides many gains in versatility and platform choice, as it is supported on almost all hardware platforms available today. Modern Linux distributions have gone beyond their hobbyist roots, emerging as the primary forward force in enterprise platforms today.

IBM is currently working with Prentice-Hall publishing on a comprehensive examination of UNIX-system-to-Linux migration: *Unix to Linux Porting: A Comprehensive Handbook*. This book focuses on porting applications from commercial UNIX systems, with select information on the Solaris OS operating system. Publication is currently planned for May of 2006, and the ISBN is: 0-13-187109-9.

END OF DOCUMENT