Preparing for the 64 bit Future of IRIX

System Software Marketing, Computer Systems Group

This note discusses the intended future directions of IRIX to accommodate 64 bit programs. It is targeted at developers to aid in their migration planning.

Executive Summary

The 64 bit version of IRIX is under development. This system will fully support today's 32 bit binaries without change, and will permit the creation of binaries which use the 64 bit features of the MIPS R4000 family of processors (R4000, R4400) and of the new Streaming Superstar RISC family.

The 64 bit version of IRIX will support a programming model in which the *long* and *long long* data types and all pointers are 64 bits. Although there are a number of other possible models, this strategy was chosen as the best overall choice. The programming model decision has ramifications in the definition of library interfaces, e.g. all pointers are 64 bits. The migration to 64 bit provides the opportunity to increase the limits on file system and individual file size. The 64 bit version of IRIX will support files and file systems of over 1 TByte.

Application developers have unparalleled flexibility in their options under the 64 bit version of IRIX. Since 32 bit binaries are fully supported, the application developer is not forced to migrate to 64 bit environments. Applications which can take advantage of 64 bit features such as 64 bit logical operations, very large virtual address spaces or very large files should consider migration.

If migration is planned, preparation can begin immediately by ensuring that modern programming practices are followed, such as strict use of ANSI C function prototypes and using system-supplied type definitions. Another key preparation step is the definition of file data structures and protocol streams which are independent of the word size. When the 64 bit version of IRIX is available, the migration can be completed in most cases by simply re-compiling in the new environment.

What does “64 bit support” Mean?

The term “64 bit” is a broad statement with a great deal of difference in the details. The general idea is that (at least) pointers are 64 bits so the virtual address space of a process can (potentially) be 2^{64} instead of 2^{32} as in current systems. The differences are in what else is 64 bits. Debates continue to rage in the community about exactly what data types are 64 bits and what stay 32 bits. Table 1 summarizes the contending proposals and also shows the current 32 bit environment. The current IRIX environment (and other 32 bit UNIX® environments) support the sizes shown in the first column. The 64 bit version of IRIX will support the LP64 model. For reference, the column labeled PC is the environment for INTEL x86 family members.

TABLE 1. C Data Type Models

C Data Type	current IRIX 32 bit	64 bit IRIX LP64	LLP64	ILP64	PC
char	8	8	8	8	8
short	16	16	16	16	16
int	32	32	32	64	16
long int	32	64	32	64	32
long long int	64	64	64	64	N/A
pointer	32	64	64	64	16/32

The column labels for the three contending 64 bit strategies indicate what primitive data types are 64 bit and what types are 32 bit. LP64 indicates that *long int, long long int* and pointers are 64 bits. LLP64 indicates that *long long int* and pointers are 64 bits. ILP64 indicates *int*, *long int*, *long long int* and pointers are 64 bits.

As stated above, the 64 bit version of IRIX will utilize the LP64 model. This choice was made after a great many (occasionally heated) discussions about the merits of the various models. The analysis leading to this choice focused on five major areas:

- compatibility with existing code, i.e. what model required the least amount of change in existing programs compiled in 64 bit
- interoperability with 32 bit processes
- functionality, i.e. does the proposed model provide the necessary functionality to exploit 64 bit processors
- standards, i.e. does the model meet existing and emerging standards
- performance

Generally, LP64 was either the first choice or a strong second in each of these areas. In no cases did the LP64 model offer the worst choice of the three contenders.

What is Required to Support 64 bit?

There are three fundamental requirements for 64 bit support:

- a processor which supports 64 bit data types efficiently. This implies that all the primary data paths on the processor chip and external to it are (at least) 64 bits wide.
- a compilation system which can produce 64 bit executables
- an operating system environment that can run 64 bit programs

The first two of these are available today on SGI products. The R4000 processors found in current SGI systems such as the INDIGO R4000 and Crimson are fully 64 bit internally and externally. The current versions of the compilation system have some of the basic elements of 64 bit support, and have been used to generate test programs for the R4000 hardware. (The released versions support only assembly language 64 bit programming.) The only missing piece, then, is the 64 bit extension to IRIX, SGI's multitheaded SMP version of the UNIX operating system.

IRIX Roadmap

Library Interfaces

Once these basic types have been defined, the library interfaces follow naturally. At the simplest level, all addresses are now 64 bits. This applies both to subroutine arguments and to addresses contained in structures which are passed by reference.

Large File and File System Support

The availability of efficient 64 bit support permits other enhancements to be made. Perhaps the most important of these is the support for files which are larger than the current limit of 2 Gbytes. (The limit is 2 Gbytes because the offset is passed as a signed 32 bit quantity. Current versions of IRIX allow file systems to be up to 8 GBytes.) In the 64 bit version of IRIX, it is planned to permit individual files and file systems of up to 2^{40} Bytes (approximately 1.2 Terabytes). This extension impacts the system calls and libraries where either the size of a file (e.g. the stat() system call) or the offset in an open file (e.g. lseek()) is used. The underlying type for the size of the file and for offsets in it becomes a *long long*. The 64 bit version of IRIX will retain the same application programming interface (API) found in the current 32 bit environments. These extensions will be supported by altering the system-defined type definitions when a program is compiled for the 64 bit environment. This "API transparency" eases the burden on the application developer in migrating to the 64 bit environment.

Availability of 64 bit IRIX

Initially, the 64 bit version of IRIX will be available only on systems which are built around the Streaming Superstar RISC processor. Future releases will be available on selected other R4000/R4400 based systems during the first year after this initial release.

There are no plans to provide 64 bit capability on older systems which utilize the R3000 processor.

64 bit Compilation

Obviously, the 64 bit support requires substantial changes to the compilation system as well as the operating system. The compilation system will be extended to support the full LP64 model discussed above. Among the extensions are the definition of a new object module format for 64 bit programs, ELF-64, and extensions for its support in all the tools which work with object modules. ELF-64 is a 64 bit version of the Extended Linking Format (ELF) used in IRIX 5.0 and other System V Release 4 systems.

The 64 bit environment requires a complete set of libraries also constructed in 64 bit (as well as continuing to provide the 32 bit versions). Developers who create their own libraries will also need to provide 64 bit versions of them. The new loader will prevent the incorrect linking of incompatible (64 bit vs. 32 bit) libraries. There will be changes in existing math libraries to take advantage of the features in the Streaming Superstar RISC processor. These will generally be transparent to the application. There are plans for additional math and vector libraries, but the details have not been completely defined.

Open Topics

A number of issues arise in migrating to 64 bits which SGI is currently working towards resolution. Most of these stem from the possibility of files bigger than 2 GBytes. To support large files, the definition or semantics of the library interfaces must be changed. The general philosophy is to attempt to minimize the impact on existing application code, and to explicitly fail rather than give misleading or incorrect results on a library call. It is our intent that 32 bit programs can access the new large files in much the same way that they access current files. Specifically, we intend to avoid the requirement for wholesale change to every 32 bit program to accommodate the new large files. Naturally, where a 32 bit program examines the size of a file it must be prepared to get a number back which is larger than 32 bits. The following are all areas which will experience some level of change:

- the stat family of system calls
- the lseek system call
- the ftell() and fgetpos() routines in the stdio library
- new options for printf formats
- backup utilities. The internal structures written to backup media currently permit only 2 GByte files. These will be extended to permit support of large files.
- transfer utilities (tar, cpio). New conventions will be established to permit large files to be supported.
- external protocol support (ftp, rcp,...). Individual programs will be altered as need be to support large files. In some cases, SGI will work with other interested firms to define standard mechanisms for support of large files.
- NFS. The offset is explicit in each NFS request and is a 32 bit quantity. Sun has proposed 64 bit extensions to NFS.

Although these areas will experience some level of change, most applications migrating to 64 bits will not require extensive change to accommodate them.

Should I Convert to 64 Bit?

The first, and probably most important, message is that you are not required to convert applications to 64 bit. Unlike some other 64 bit implementations, the 64 bit version of IRIX will fully support today's 32 bit binaries. These current binaries can run without change on both R4000 family and R3000 family processors. These binaries can also run on Streaming Superstar RISC systems without change. Only if the application can benefit from the features and performance available to 64 bit programs, must it be converted.

• *Happy with 32 bit?*

Do nothing, 32 bit works fine. Have a nice day.

• *Need 64 bit features?*

Start today to clean up the code to modern practices. Recompile on IRIX-64 and you're done.

• *Coming from a supercomputer?*

Wait for IRIX-64 and use compatibility support for Cray and Convex.

Basic Conversion Considerations

In making a choice of whether to convert to 64 bits or not, the following should be considered:

- can the application benefit from 64 bit logical and/or arithmetic operations?
- can the application benefit from a virtual address space larger than 32 bits?
- can the application benefit from files larger than 32 bits?
- is the application being ported from a 64 bit (supercomputer) environment?

Obviously this is not an exhaustive list, but it touches on some of the strongest reasons to consider the possibility of 64 bit migration.

64 bit Arithmetic and Logical Operation Speedups

The ability to do logical operations in 64 bit strides can potentially double the speed of critical inner loops in applications such as image processing or ECAD circuit simulations where bits in vectors or matrices are logically manipulated. To exploit this capability, generally very minor surgery to the program is required. Simple changes to the declarations of the data structures and alteration of loop bounds will usually suffice.

The efficiency of supporting 64 bit integer arithmetic is also potentially more than doubled in 64 bit operation (as compared to doing the same operation in a 32 bit environment). As with the logical operations, some change to the software is required to exploit this capability, but it will be typically quite localized.

Large Address Spaces

Applications are beginning to require more than 32 bits of addressability to achieve their required performance or functionality. For example, a data base system may wish to employ a very large page cache to improve system performance. In some of today's CAE environments, the 32 bit limitation constrains the size of the system being modeled. Removing this limitation can permit larger, more complex systems to be modeled. Application developers should be aware that there are secondary costs associated with the decision to migrate to 64 bits. Notably, the storage for pointers is doubled, even if no other 64 bit features are used. This can affect finely tuned applications by increasing the size of certain structures so they no longer "fit" in, say, a page frame. These disadvantages may well be offset by some of the secondary benefits of conversion to 64 bits. The most important of these are:

- improved register utilization, especially from the additional floating point registers available in 64 bit mode
- improved subroutine argument passing
- improved memory to memory move speed

Large Files

Some applications, such as geophysics and particle physics, may require access to files which have a natural size greater than 32 bits (2 GBytes, since the offset is signed). One case where this is common is when the application must interact with supercomputer systems, which permit such large files. The 64 bit version of IRIX will support files of up to 1 TByte.

Super Computer Migration

This file size issue is one special case of the general issue of porting applications from other 64 bit supercomputer environments. Such applications are already adapted to 64 bits so are likely to need only minor change. (Indeed, reverting to 32 bits as is required by vendors whose systems are not 64 bit capable, such as Sun, HP and IBM, can be even more challenging than migrating to 64 bits from a 32 bit environment.) To aid in the migration, the compilation system will have optional features to provide source code compatibility, compatible compiler options, and compatible libraries with the Cray and Convex environments.

What Should I do Today?

Much of the preparation for 64 bit migration is simply employing good programming practices, such as the use of ANSI C function prototype declarations, and type definitions for the declaration of storage. Properly employed, these techniques permit a single source to be compiled for either 32 or 64 bit operation. Program style checkers such as lint can also be employed to help find potential trouble spots. SGI will provide a checker which is specifically tuned to identifying potential problems in 64 bit migration, such as questionable casts. This tool has been successfully used in the conversion of the IRIX kernel to the base for the 64 bit version of IRIX.

Don't Assume Data Type Sizes

The basic requirement is not to make assumptions about the absolute or relative size of *int*, *long* and pointer data types. It is sometimes the case that in older programs these assumptions are subtly built into the code, and the code will require modification to operate correctly in a 64 bit environment. In a 64 bit environment, the sizes of these data types differ, so they cannot be as freely interchanged as they can be in a 32 bit environment.

Probably the most commonly occurring problem is the assumption that *sizeof(int) == sizeof (void *)*. This can occur explicitly as in the case of a cast of a pointer to an *int*, or implicitly as a pointer is passed as an actual argument to a procedure where the formal argument is declared to be an *int*. It is also a common (bad) practice to use an *int* as an implicit union in structure declarations where either an *int* or a pointer may occur. In all of these cases, truncation of the high order 32 bits of a 64 bit address may occur. Careful attention to the warnings produced by the compiler about casts and rigorous use of ANSI C function prototypes can eliminate the problems between actual parameters and the formal declaration. In existing code, be especially careful about casts which have been inserted to "shut up" the warnings. These warnings are a nuisance in 32 bit environments, but in 64 bit environments, they represent impending disaster.

Pointer Arithmetic

A more subtle problem occurs when two pointers are subtracted to produce an offset. This will be a 64 bit quantity in a 64 bit environment, not an *int*. The proper declaration for such a result is the type definition *ptrdiff_t* from stddef.h. This type definition permits the program to work in either a 64 bit or 32 bit environment without change to the source.

Improper Union Use

Careful attention must be paid to union declarations which are used to access the structure of one of the basic types, e.g. the union of a four character array with a long. This has always been a bad practice due to endian issues, but it is much worse in a 64 bit environment. This practice is sometimes found when externally generated data such as protocol streams must be mapped into local constants. For example, a protocol header might be declared to be a union of a character array and a structure which contains the fields of the header. Although this style would have been non-portable from byte ordering (endian) issues, it may be used on code not intended to be ported to other architectures. Such a style will encounter problems in a 64 bit environment. Generally, shifts and masks must be used to extract fields from such streams.

Extensible Constants

The declaration of constants with the high order bit set may not produce the expected results if they are 64 bit quantities. For example 0xFFFFFFFF is just a large 64 bit integer, not -1 in a 64 bit environment. The recommended practice has long been to use extensible constants, but many programmers have not always followed the recommendation. This same type of problem occurs with declarations of bit masks for logical operation.

Hard to Find Problems

Some bad practices cannot be found easily by automated tools. For example, it is (unfortunately) not uncommon to see memory being allocated with an assumption about the size of the primitive object, e.g. malloc(4*count). Were the object in question one which was changing to 64 bits (8 bytes), the assumption would no longer be valid, and the program would be in error. Although the sophisticated CASE environments provided by CASEvision/WorkShop™ can be extremely helpful in finding these types of errors, they cannot substitute for human intervention and insight.

Run Time Issues

All of the above discussion has been on static issues. There are also potential pitfalls in the dynamics of the program operation. Migration to 64 bit may expose latent errors in the program such as overflows in arithmetic and shifts. These may not occur in the 64 bit environment, and if they were intended, appropriate mask operations must be performed. Similar problems can occur with sign or zero extension.

Machine Independent Data Files and Protocols

One final issue is the need to plan for data files and external protocol exchanges which are insensitive to the 32/64 bit issue. The systems will have ample processor power with which to support machine independent representations and to convert to the appropriate internal form as need be.

What do I do when I have IRIX-64?

The specific intent is that a properly constructed source base can be compiled for either 64 bit or 32 bit without change. So, if you have prepared well, all that you need to do is recompile.

The unparalleled flexibility of the MIPS R4000 family and Streaming Superstar RISC architectures permit you to defer the actual migration to a 64 bit environment until a time of your choice. Naturally, your software won't be able to take complete advantage of the 64 bit hardware features until it migrates to 64 bit. In many applications, only some programs must be converted to 64 bit, with the rest remaining 32 bit. For example, it is unlikely that one would need to convert many of the components "on the edges" of large applications, such as installation and setup tools, simple input tools, etc.