The Mandate of Application Compatibility in SGI IRIX 6.5

Abstract

In May 1998, SGI released IRIX 6.5, its UNIX-based operating system. Each quarter SGI releases a new minor version, IRIX 6.5.X. These quarterly releases are rigorously tested to ensure reliability and strict compatibility both up and down the release stream. The IRIX 6.5 release stream design allows applications to run correctly on any version of IRIX 6.5, regardless of the version on which the application was developed and tested.

This white paper describes the application compatibility protections offered by IRIX 6.5. These protective attributes represent investment protection for customers and application providers. Because of them, a customer can upgrade to any release of the IRIX 6.5 release stream and be confident that its applications will continue to work properly. Likewise, an application provider can develop its products on any release and easily support those products across its entire customer base. The customer can be running any version within the IRIX 6.5 release stream.

This paper begins with a general introduction to application compatibility issues and then addresses in depth the IRIX 6.5 application compatibility and quality assurance efforts. A detailed reference section closes this paper.

Introduction

Applications do not exist in a vacuum. An application provider typically develops an application on a particular computer system running a particular operating system. However, the user usually runs that application on a diverse array of computer systems that are usually running different versions of the operating system than that on which the application was developed.

What allows this to work is system and software “compatibility.” That is, the computer system on which the application was developed and the one on which it is run are “compatible” with respect to applications. Generally, operating systems from different manufacturers are incompatible with one another. Indeed, sometimes computer systems from the same manufacturer may be incompatible. (This latter situation usually occurs only during major product transition periods.)

The job of an application provider is to create an application that solves a particular problem for users. However, supporting an application on each “compatible platform”

requires the application provider to invest considerable time and effort. Ideally, the provider wants to reach the greatest number of users while supporting the fewest number of platforms.

The typical computer system manufacturer recognizes this “market share” problem and generally strives to maintain compatibility across both its product lines and multiple releases of its operating system software. This effort is in its own best interest as every change that results in incompatibility will produce, at minimum, a temporary but costly lack of application coverage, as the application providers struggle to resolve the incompatibility.

Factors That Govern Application Compatibility

There are several, mostly orthogonal, compatibility issues that affect applications, as follows:

- **Compatibility of syntax, semantics, and binary interfaces of all bundled APIs and their associated data structures**

  An application programming interface (API) is used by an application to request services from the underlying operating system and supporting libraries. Issues include the names and locations of header files, the names and existence of symbols and interfaces, the number, order, and data type of interface parameters, the return value types of functions, and all of the subtle defined semantics of interfaces and global symbols.

  For example, if one manufacturer provides a service entry point named “foo” that requires four input parameters, and another manufacturer supplies a version of “foo” that requires five input parameters, these definitions of “foo” are incompatible.

- **Application executable file format compatibility**

  The file format in which an application is stored is generally a binary format that the operating system recognizes as an “executable image.”

- **Operating system and library ABI compatibility**

  This concerns the application binary interfaces (ABIs) and conventions used within an application, the supporting operating system software, the CPU registers used to pass input parameters and return results, the exact binary format of various fundamental data structures, and so on.

- **CPU ISA standards and compatibility**

  This concerns the Instruction Set Architecture (ISA) standards that the CPU recognizes, and their exact semantics.

- **Dependencies on special hardware and its capabilities**

  This involves the features supported by particular hardware devices and their software drivers. An application may depend on hardware functionality available only on a specific system or optional product, such as a digital video card. Other examples include the necessity of ccNUMA hardware, graphics-adapter-specific OpenGL extensions, and so on.

  Often these dependencies can be detected at run-time. A well-written application tests for the presence of special hardware and capabilities and either changes its behavior, degrades, or fails gracefully.

If the executable object file format, the ABI, or the ISA differs between one computer or operating system and another, an application will need to be recompiled in order to be compatible with the other system. If the API changes, portions of the application will need to be rewritten in order to accommodate the changes. Thus, changes to APIs are especially unpleasant. If special hardware is used with different capabilities, fairly extensive changes to the application will be necessary.

A prudent computer system manufacturer is extremely cautious when changing any of the foregoing because of the potential impact to the availability of applications for the new system. This is so important that most manufacturers bend over backward to seamlessly support as many old interfaces as possible and consistently choose to add new functionality rather than change old functionality. This is known as “backward compatibility,” which means that the newer operating system release is “backward compatible” with applications developed on older releases.

However, adding new functionality while leaving old functionality intact also has its downside. For example, an application developed on a new system may not run on an older system. This factor often causes an application provider to develop its products on the “lowest common denominator” of a computer system manufacturer’s product line. That is, if a manufacturer is currently supporting three different “backward compatible” versions of its products, an application provider may develop its application on the oldest version in order to offer that application across the entire line of the three supported systems.

One method of tackling this lowest common denominator problem is to ensure that applications developed on newer releases of an operating system can also run on older releases. This is referred to as “forward compatibility,” which means that the older operating system releases are “forward compatible” with applications developed on newer releases. This requires that all of the issues concerning application compatibility be carefully monitored and tested as new operating system releases are developed in order to ensure that no change will result in an incompatibility with the older releases. This also requires that any new operating system interfaces be added in a controlled manner in order to allow applications using these new interfaces to recover gracefully when they are run on older releases without them.

Because API differences are so problematic, international standards for APIs have been developed to govern many of the interfaces that applications use. Most mainstream computer system manufacturers have adopted these standards. Examples of these standards include ANSI C, POSIX, X/Open, the X Window System, and so on. They govern API compatibility issues regarding language, operating system, libraries, and so on.

Application Dependencies That Affect Compatibility

Other issues, beyond those listed previously, can affect the ability of an application to run correctly—even on computer systems that offer otherwise compatible interfaces, facilities, and features. These issues include the following:

- Dependencies on undocumented features or even operating system bugs. A "conforming" application adheres to the documented interfaces and their rules.
- Dependencies on operating system bug fixes that are available only in newer operating system releases. An application provider usually tests its applications on that set of operating system releases on which it is willing to support its customers. Operating system vendors like SGI expend great efforts to ensure that each release fixes problems identified by customers but that the release does not, itself, introduce new problems.
- Dependencies on certain system capacities (e.g., amount of memory), system features (e.g., system page size), or system tuning parameters (e.g., maximum number of allowed processes). It is the responsibility of the application provider to document all such dependencies.
- Bugs within the application itself that become apparent only when run on certain types of systems (e.g., timing dependencies, or bugs within a multi-threaded application that appear only on multi-processor systems). It is the responsibility of the application provider to understand its application, identify those areas in which bugs might be found, and design appropriate testing to cover these areas.

All of these issues usually require the application provider to conduct tests on diverse computer systems and operating system releases in order to offer support for its applications on those systems and releases.

This ongoing testing burden is minimized by writing applications that adhere to the documented interfaces and their usage rules, designing good application tests to cover likely problem areas, and documenting all system dependencies. Such properly written and tested applications are referred to as "conforming applications." A conforming application achieves the greatest level of compatibility. (The testing burden is also minimized if the operating system vendor uses a well-designed release model that avoids interface incompatibilities and focuses on producing a stream of reliable, regression-free releases.)

IRIX 6.5 Application Compatibility Design

As have many computer system manufacturers, SGI has introduced new compatibility standards for its computer systems and IRIX. Some of these standards were driven by the need for functionality and performance and some were driven by the need to comply with international standards.

In almost all cases, these changes involved carefully adding new functionality while continuing to support legacy interfaces. That is, applications developed on older versions of IRIX continued to run correctly on newer versions of IRIX. Still, these changes were problematic because application providers needed to develop on the lowest common denominator.

With the release of IRIX 6.5 in May 1998, SGI established a new compatibility standard, again backward compatible with applications developed on older versions of IRIX. However, unlike older versions of IRIX, IRIX 6.5 established a new mechanism for releasing new versions of IRIX 6.5 that offers application compatibility across all releases of IRIX 6.5 and the computer systems supported by IRIX 6.5. This compatibility design and the new release mechanism were driven by the following application provider and customer principles:

- New features, new computer systems, performance enhancements, and bug fixes are good.
- Compatibility changes of any kind are bad.
- Reliability problems of any kind are bad.

Specifically, the IRIX 6.5 release stream offers application providers three release assurances: constant and extensive product improvements, assurance of compatibility, and assurance of reliability. These assurances are discussed in the following sections:

- **Constant and extensive product improvements**

  Each release of IRIX 6.5.X includes support for new customer-requested features, new hardware, performance enhancements, and bug fixes. All of this work is performed under the overriding mandates of compatibility and reliability. All changes are reviewed and carefully tested before being integrated into the IRIX 6.5 release stream. Each release of IRIX 6.5.X includes a document entitled Start Here, which includes a complete list of all the major work in that release.

- **Assurance of compatibility**

  The IRIX 6.5 release series is, and always will be, an "all platforms" release. That is, each release of IRIX 6.5.X contains support for all the system platforms supported by IRIX 6.5 and any new system platforms introduced during the intervening time. As new IRIX/MIPS system platforms are released, each is smoothly integrated into the IRIX 6.5.X release series. There are no "special" versions of IRIX for new system platforms. Good examples of this are the releases of the SGI Origin 3000 series and the SGI Onyx 3000 series in IRIX 6.5.9.

  The IRIX 6.5 release series offers binary compatibility up and down the release stream, across the maintenance and feature branches of the release stream, and across all systems. That is, a conforming application may be compiled on any version of IRIX 6.5.X, on either the maintenance or feature branch, and on any system. The resulting application binary will then run on any other version of IRIX 6.5.X, on either the maintenance or feature branch, and on any other system.

  The following two sections discuss two circumstances under which this obviously cannot hold true: if special hardware is needed, and if new interfaces, introduced after the base release, are used.

  If special hardware is needed. As noted previously, if an application depends on particular hardware features, it must be run on systems with that hardware. There are two special cases of hardware dependencies:

  - Because of hardware limitations, some of the desktop workstations supported by IRIX 6.5 cannot run 64-bit applications. These are the Indigo, Indy, Indigo2/R4000/R5000, and O2.

- Some older legacy systems supported by IRIX 6.5 use MIPS R4000-based CPUs that can only execute instructions from the MIPS III ISA. Most systems supported by IRIX 6.5 contain CPUs that conform to the MIPS IV ISA, which is a strict superset of the MIPS III ISA. Attempts to execute new CPU instructions introduced in the MIPS IV ISA on systems with R4000-based CPUs will result in the application aborting with a reserved instruction fault. These legacy MIPS III ISA systems are the Challenge/R4000, Indigo, Indy/R4000 and Indigo2/R4000.

If new interfaces, introduced after the base release, are used. If an application makes use of a new, customer-requested software feature introduced after the base IRIX 6.5 release, that application must be run on a version of IRIX 6.5.X that offers that feature. However, SGI has provided application providers with simple methods to work around even this kind of problem via the "optional interface facility." This allows an application to test for the presence of an interface and either degrade or exit gracefully if the interface is not present on the currently installed release of IRIX 6.5.X. For example, an application that makes use of the new job limits features introduced in IRIX 6.5.7f may contain code like the following. For more information on this facility, see the online man page optionalsym(1).

```
if (_MIPS_SYMBOL_PRESENT(getjid) == 0) {
    fprintf(stderr,
             "This application uses Job Limits facilities and thus\n "
             "can only be run under IRIX 6.5.7f or later.
            Sorry.\n ");
    exit(EXIT_FAILURE);
}
```

If an application makes use of a new dynamic shared library (DSO), it must be run on a version of IRIX 6.5.X with that DSO. In this case, run-time checks are often difficult to arrange in the application itself because the dynamic run-time linker (rld) detects the missing DSO before the application main line code is executed. Approaches to this problem include the following:

- Using application installation packaging that allows the application to be installed only on compatible systems.
- Using execution scripts that check for a compatible installed IRIX 6.5.X release.
- Using delay-loaded DSOs.

- **Assurance of reliability**

  With the IRIX 6.5 release series, SGI is committed to the policy of "no regressions." This is obviously a very difficult task, given that IRIX 6.5 supports a wide-ranging set of features and hardware. Nonetheless, it is our top priority.

  To help ensure that this goal is reached, extensive code reviews and testing occur throughout the development process. In particular, we run each IRIX 6.5.X release on our own internal production machines before shipping the release to customers. This practice has become known as "eating our own cooking before we serve it to our customers." The quarterly release process also provides the flexibility of waiting until changes and new features are ready before including them in the release stream.

  Of course, even with all this testing and the use of the releases on our own mission-critical servers, a very small number of regressions have slipped through during the years. Our commitment is to fix these immediately and provide these fixes in a timely manner.

The result of this compatibility design is that an application provider may develop and test an application against any release of IRIX 6.5.X without worrying about whether the resulting product will work on another version of IRIX 6.5.X or another system. It will work.

Note: For IRIX 6.5 kernel components, there is a similar compatibility design. The only difference is that backward compatibility alone is mandated. That is, if a kernel component is developed and tested against IRIX 6.5.X, that kernel component binary will work correctly on any release of IRIX 6.5.X that is the same or newer than the release on which the kernel component was developed. The reason for this restriction is that it is impossible to verify that a kernel component does not depend on some kernel facility that may not be present in an older release. As with applications, it is important that kernel components use only documented interfaces and their documented semantics.

Fundamentally, our goal is to make the decision of whether to upgrade to the newest release of IRIX 6.5 a completely surprise-free "no brainer." Our philosophy is that an operating system upgrade should be boring. We have proudly maintained this standard of compatibility and reliability since the original release of IRIX 6.5 in May 1998, and we look forward to maintaining this standard for many years.

Detailed Reference on IRIX 6.5 Compatibility Issues

IRIX 6.5 compatibility issues are explained in detail in the following sections:

- API compatibility in IRIX 6.5.
- Application executable file formats supported by IRIX 6.5.
- ABIs supported by IRIX 6.5.
- ISAs supported by IRIX 6.5.

API compatibility in IRIX 6.5

The IRIX 6.5 release series offers binary and API compatibility forward, backward, between the maintenance and feature branches of the release stream, and across all systems. That is, a conforming application may be developed on any version of IRIX 6.5.X, on either the maintenance or feature branch, and on any system. The resulting application will run on any other version of IRIX 6.5.X, on either the maintenance or feature branch, and on any system.

Some new customer-requested interfaces have been added to IRIX 6.5 across the various releases. IRIX 6.5 provides an "optional interface" facility, _MIPS_SYMBOL_PRESENT(), which allows an application to determine if a new interface is available on the currently running system. If not, the facility allows the application to either provide degraded functionality or exit gracefully.

IRIX 6.5 is compliant with a number of international API standards, including ANSI C, POSIX 1003.4, and X/Open 1995.

Application executable file formats supported by IRIX 6.5

IRIX 6.5 supports ELF (Executable and Linking Format), a standard executable and object file format used by most UNIX-based and UNIX-like operating systems. Support for the older COFF (Common Object File Format) was dropped in IRIX 6.2.

ABIs supported by IRIX 6.5

IRIX 6.5 supports two ABIs under active development and one older legacy ABI that is maintained for compatibility with legacy applications, as follows:

- **32.** The "old 32-bit" ABI (sometimes referred to as "O32") was the original 32-bit ABI introduced when the first MIPS CPUs were developed. The O32 ABI supports only the MIPS I and MIPS II ISAs. The O32 ABI is supported for legacy application binaries.
- **N32.** The "new 32-bit" ABI was released in July 1995 in conjunction with the release of IRIX 6.1. The N32 ABI was developed to address several shortcomings in the performance of the O32 ABI and also to accommodate the larger CPU register sets of the MIPS III ISA. N32 is the ABI of choice for applications that do not need large virtual address spaces.
- **64.** The "64-bit" ABI (sometimes referred to as "N64") was released in August 1994 in conjunction with the release of IRIX 6.0, the first 64-bit IRIX operating system. The 64-bit ABI was developed to enable applications to use virtual address spaces larger than two gigabytes (2^{32}). The 64-bit ABI is identical to the N32 ABI in register-calling conventions and other ABI issues, except that "long" integers and pointers are 64 bits in length. (In the N32 ABI, these are 32 bits in length.) This promotion of the "long" integer and pointer types to 64 bits is often referred to as the LP64 model.

All of the ABIs offer a common API. Thus, it is usually a simple matter of recompilation to convert a correctly written application to use a different ABI. The compilers for the N32 and 64-bit ABIs are more stringent with regard to programming language standards and code consistency. This may result in some compiler warnings and errors when compiling an application that was formerly compiled for the O32 ABI. If an application contains coding errors, such as an assumption that integers and pointers are the same size, then work is required to make the application "64-bit clean." The N32 and N64 compilers issue diagnostics to search for such coding errors.

All computer systems supported by IRIX 6.5 support the 32-bit ABIs, and most also support the 64-bit ABI. Some of the desktop workstations supported by IRIX 6.5 cannot run 64-bit applications because of hardware limitations. These are the Indigo, Indy, Indigo2/R4000/R5000, and O2. All future IRIX/MIPS desktop and server system products will support 64-bit applications.

For more detailed information on these ABIs, see the online man page `ABI(5)`.

ISAs supported by IRIX 6.5

IRIX 6.5 supports four MIPS ISAs: MIPS I, II, III, and IV. Each MIPS ISA is a strict superset of the preceding MIPS ISAs. The MIPS III ISA was released in conjunction with the release of the R4000 CPU family from MIPS Technologies, Inc., in 1992. The MIPS IV ISA was released in conjunction with the R5000 CPU family from MIPS, and the R8000 and R10000 CPUs from SGI in 1994.

All systems supported by IRIX 6.5 support the MIPS III ISA, and the majority support the MIPS IV ISA. Systems that do not support the MIPS IV ISA are older legacy systems that use MIPS R4000-based CPUs. All currently shipping IRIX/MIPS systems support the MIPS IV ISA, as will all future SGI IRIX/MIPS systems.