Constructing MIPS® ABI Compliant Binaries on the IRIX™ 5.x Operating System

E.J. McCauley

This note discusses how to build a binary that complies with the MIPS ABI, and is, therefore portable to all member systems. This is preliminary information, so feedback on errors or omissions is desired. Send mail to mccauley@sgi.com. flames > /dev/null

Introduction

The MIPS ABI group is an organization comprising the vendors of computer systems based on the MIPS processor family and which utilize the UNIX Systems Laboratories System V Release 4 version of the UNIX™ operating system. The primary purpose of the MIPS ABI group is to encourage the development of applications for these systems, by making it possible for a suitably constructed binary program to execute without changes on all member platforms. The mechanism for this interoperability is to ensure that the program complies with the MIPS Processor Supplement to the System V Release 4 Applications Binary Interface ("the MIPS ABI"). Such programs will be referred to as ABI compliant programs in this document.

Each of the member companies has devoted significant technical resources to making sure that the myriad of details are all resolved in ways which permit binary interoperability. The work has included verification of interoperability with both test suites and actual large scale commercial applications. The first of these applications are now commercially available.

Generation of ABI compliant binaries is basically simple, but there are a number of small traps for the unwary. This note attempts to cover the basic steps and to point out the most common errors.

The model for the MIPS ABI group is based on the notion of a Reference Platform (RP). The software developer builds and tests the application on the RP, and the other members platforms guarantee that ABI compliant executables that operate correctly on the RP will also operate correctly on their platforms. This model is that of the Personal Computer marketplace.

ABI Definition

The MIPS ABI is defined by a series of documents. (There is a annotated bibliography at the end of this document.) The basic defining document is the System V Release 4 Interface Definition, Issue 3 (SVID 3). The System V Applications Binary Interface ("Generic ABI" or gABI) defines those elements of SVID 3 that are expected to be available on all ABI compliant systems, regardless of processor type. The System V Applications Binary Interface MIPS Processor Supplement ("Processor Specific ABI", psABI) defines how the gABI is to be implemented on the MIPS processor family. The MIPS Processor ABI Conformance Guide ("the Black Book" from the color of the cover of the first distributed version) defines additional interfaces agreed upon by the MIPS ABI group members, and offers guidance to the system developer of ABI compliant systems. The Black Book is available through SGI Developer Support.

An ABI compliant program is one which adheres the ABI-defined interfaces, and which has been constructed to comply with the formats and conventions for ABI executables. All ABI compliant programs use Dynamic Shared Objects (DSOs) exclusively to access system services. DSO support in the IRIX 5.x environment is available only for SVR4 executables in the ELF format, which is also required by the ABI. Static linking of libc would render the program non-compliant. It is the DSO mechanism which makes it possible to move a common binary among systems with significantly different internal implementations.

The ABI is defined by the above mentioned documents. The following briefly summarizes the types of interfaces available to ABI programs

- standard system services (libc)
- TLI networking (libnsl)
- X11R5 Window System (libX11). The X Toolkit, libXt, is available as a static archive that can be bound with the program at build time.
- OSF/Motif GUI (libXm), available as a static archive
- fast mutual exclusion library for process synchronization (libmutex)
- Sockets networking (libsocket)
- BSD 4.3 services (libucb)
- dynamic linking under program control (libdl)

(The standard IRIX environment includes a number of other libraries and features outside of the ABI. Programs which utilize these IRIX specific features would not be portable to other systems.)

ABI Compliant Program Development

The first step is to do the development on the RP. The MIPS ABI group has approved the Silicon Graphics IRIX™ 5.1 / IRIS™ Development Option 3.17 environment as the

RP. This environment requires a supplementary ABI Construction Kit which is available through SGI Developer Support until it is integrated in a future software release.

Any candidate ABI program must, rather obviously, restrict itself to the use of the System V Release 4 (SVR4) interfaces (including the interfaces defined in the Black Book), as they are the basis of the ABI. Systems which have been ported to SVR4 on other systems should require little, if any, changes in the source. Some source changes may be desirable to take advantage of Black Book features not available on other platform families such as the socket library or the mutual exclusion library. Programs migrating from prior versions (pre IRIX 5.x) of the IRIX operating system will require modifications to comply with the new interfaces.

The essence of ABI compliance is to make the executable in a way which leaves only ABI defined external references. These references are satisfied at run time by linking with the DSO libraries on each of the target systems. In the IRIX RP, this is accomplished by building the executable against a specially prepared set of libraries and include files located in /usr/lib/abi. (This set of libraries and include files makes up the bulk of the ABI Construction Kit.) These libraries have been structured to contain only ABI defined references, so the application developer will see warnings if interfaces outside the ABI are used.

To be ABI compliant, the application may use only the DSO libraries defined by the ABI. These are:

- libc.so.1
- libnsl.so
- libX11.so.2
- linmutex.so
- libsocket.so
- libdl.so

The following static libraries are available in the ABI build environment:

- libXm.a
- libXt.a
- libcurses.a
- libgen.a
- libm.a
- libmld.a

Applications are free to link in other static libraries in their build, and to specify DSOs that are supplied with the application. Note that the application must be linked with the DSOs from the ABI build environment in /usr/lib/abi, not the standard system libraries. On IRIX systems, the standard system libraries include many functions which are outside the scope of the ABI, so while the resulting binary would operate correctly on the IRIX system, it might not be portable to other systems.

ABI compliant programs must be built with the loader flag -abi set to inform the loader to generate the binary in the format required by the ABI. A number of include files have slight changes in them for ABI builds, so the define _ABI_SOURCE must be set.

So, each compilation must be done with the following flags set:

```
 -abi -Wl,-abi -D_ABI_SOURCE -L -L/usr/lib/abi -I/usr/include/abi
```

When the executable is linked to the libraries, it must set the following loader flags:

```
-abi -L -L/usr/lib/abi
```

There are several static archives in the ABI build environment. For example, the OSF/Motif library is static, as is the X Toolkit library. This means that components from these static libraries can be incorporated into the executable as it is built. Doing so isolates the executable from known incompatibilities among the various implementations of these libraries on different platforms.

When building an ABI compliant executable, care must be exerted to ensure that only ABI references are left unsatisfied. For example, a common error has been to omit the link with libnsl in programs using network services (i.e. no -lnsl on the link step). This leaves a few references that are not part of the ABI, but will operate correctly on IRIX. The failures occur when the executable is moved to another system.

A key tool for catching the types of errors in the last paragraph is *check_abi_compliance*. This tool analyzes the dangling references in an executable and compares them against lists of the references available in the ABI. It reports references that are outside the ABI, and also missing references in the executable that are expected by the ABI libraries and run time environment. It is not infallible, and cannot tell whether the reference is actually used in the execution of the program. Still, it's the best tool available for checking that the binary is ABI compliant. (This program is part of the ABI Construction Kit.)

The whole area of ABI applications compliance tests is a very active one, so stay tuned for developments.

Some other useful tools are:

- *check_for_syscalls* - looks at the executable for system calls, which are not allowed in ABI executables. (All the system calls are made by the DSOs. This is a key requirement for portability.)
- *elfdump* - dumps information about an ELF file.

After checking the executable, the next steps are standard types of testing on the RP. For the early ports, we recommend at least sanity testing on all platforms to be supported. As confidence in the ABI and the RP is gained, this will rapidly become unnecessary.

Many applications include libraries as part of their package. This is fairly straightforward, and is vastly simpler than to older SVR3 static shared library scheme found in the IRIX 4.0.5 environment (and others). All ABI programs must use Position Independent

Code (PIC) for calls to DSOs, and the DSO library must also follow PIC conventions at the call point. (Current IRIX compilers will generate the entire executable as PIC.) In the IRIX 5.x environment, PIC is the default style. For assembly language programs, the IRIS Development Option manuals describe PIC coding.

Packaging

There are two basic options for packaging the resulting ABI port for delivery:

- use the SVR4 packaging tools (*pkgmake*, *pkgtrans*, etc.). These are guaranteed to be available on all ABI compliant systems.
- use your own packaging and installation tools. These must be restricted to using the programs defined in Chapter 8 of gABI. Vendors must assure that these programs all behave in the standard SVR4 ways (arguments, flags, syntax, semantics). There is a great deal of variability among the member vendors systems, so some care must be exerted in using your own packaging tools.

Although a number of media options are discussed in the Black Book, the only one common to all systems is QIC-150 cartridge tape. It is reasonable to assume that CDROM will become increasingly common, but it is not yet universally available.

Note that ABI applications do not use the IRIX *inst* format.

Special Situations / Potential Pitfalls

Many applications include some sort of construction kit which is used by the end-user to construct tailored applications, e.g. the 4GL environments commonly found in DBMS. If such environments use only DSOs, there are generally no problems. However, some of these construction kits must use static archives. Static archives (libraries) that are distributed with the product pose some challenges. One set of problems is fairly straightforward, but still vexing; linking with static archives on the target systems (other than the RP) usually involves different arguments and flags on each system. This is a reflection of the differences in heritage of the compilers on these systems. To ease the problem, the vendors have agreed to provide a "standard" complier wrappers, *abicc* and *abild* that interpret the arguments and flags from the SVR4 Software Generation System (SGS) compilers and map them into the local arguments and flags. Note that not all of the SGS arguments and flags can be mapped. It is safest to restrict the use of flags and arguments to those defined in the SVID 3.0. Such applications should be very careful in their assumptions about target system library contents and interfaces.

Another potential library linking problem is that the libraries and include files outside the ABI are not guaranteed to be identical on all systems.

ABI applications are permitted to generate code and transfer to it. They must ensure that caches are flushed to memory so that, in particular, the I-cache does not contain the prior contents of the memory cells with the newly generated program in them. This cache flushing can be done with the ABI defined mprotect() routine, by changing the protection of the relevant page(s) to be executable. Such applications should be aware of the errata sheets for the R4000. There are known problems with branch instructions and divide instructions which occur at the end of a page and which encounter a TLB miss for the next page. The loader and the operating system attempt to control these, but if the application, say, generates code and transfers to it, it must make sure to avoid these operations.

At present, the ABI is defined only for C. Work on definition of ABI interfaces for Fortran (beyond those of the C runtime library), C++ and other languages is on-going. Fortran subroutines that use only ABI facilities are permitted, but this is rather restrictive.

The architecture of the Global Offset Table (GOT) is such that a single library may contain only 16K external symbols. There is no work around for this beyond breaking the single library into smaller pieces. The MIPS ABI group is working on extensions to the ABI for much larger GOT, but it will be some time before these proposals are approved and are available on all vendor’s systems.

Annotated Bibliography

System V Applications Binary Interface, Revised Edition, UNIX Press (Prentice Hall), 1990, ISBN 0-13-880410-9. Defines the Generic ABI, mostly by reference to SVID 3.

System V Application Binary Interface, MIPS Processor Supplement, UNIX Press (Prentice Hall) 1990, ISBN 0-13-880170-3. Defines the MIPS processor implementation of the Generic ABI. A revision is planned. The changes are largely clarifications which come as a result of the MIPS ABI Group experience in making independently implemented systems binary compatible.

System V Interface Definition, Third Edition, AT&T/Prentice Hall 1989. (Five Volumes). Commonly referred to as SVID3, this set of documents define the SVR4 API.

MIPS Processor ABI Porting and Conformance Guide, available from SGI Developer Support. Also called “The Black Book”, this clarifies the implementation of the ABI and describes the extensions to the gABI agreed upon by the MIPS ABI group members.

X Window System: the Complete Reference to Xlib, X Protocol, ICCCM, XLFD, Robert W. Scheifler and James Gettys, with Jim Flowers and David Rosenthal, 3rd Edition, Digital Press, ISBN 1-55558-088-2. This is the primary X11R5 reference.

OSF/Motif Programmer’s Guide, Revision 1.2, Open Software Foundation, Prentice Hall 1992, ISBN 0-13-643107-0. These two documents are the primary OSF/Motif 1.2 references.

OSF/Motif Programmer’s Reference, Revision 1.2, Open Software Foundation, Prentice Hall 1992, ISBN 0-13-643115-1.