Compliance with SVID3 8

Introduction

System V Release 4 (SVR4) is an important effort to standardize the UNIX operating system and unify its various derivatives. Solaris 2.x, which is an environment based on SVR4, fully conforms to the *UNIX System V Interface Definition, Third Edition* (SVID3). This chapter describes how to develop source code that can be used in both the SunOS 4.1.x system and the Solaris 2.x environment.

System V Interface Definition

The System V Interface Definition (SVID) details the external interfaces to the services that are provided by all UNIX System V environments. Its purpose is to specify an operating system environment that allows users to create applications that are independent of any particular computer hardware.

As the name implies, the SVID defines the interface layers; applications are free to utilize—and platform vendors are free to implement—the SVID in any manner they choose. The criterion, however, is that in all SVID-compliant systems, commands and system calls are written using the same parameters and must demonstrate the same "behavior". The SVID specifies the runtime behavior to be seen by either the application software or an end user. The emphasis is on defining a common computing environment, not on the internals of the operating system, such as the scheduler or memory manager.

The SVID is compliant with the POSIX 1003.1 Full Use Standard and ANSI C X3J11 and will continue to evolve towards compliance with other industry standards as they are approved.

The SVID divides the components of System V into a Base System and a series of Extension Definitions. The Base System provides a minimal, stand-alone runtime environment for application programs originally written in a language such as C. The application requires only the Base System for proper implementation.

The Base System extensions provide a path that leads to a full System V configuration, as well as a mechanism for introducing new technology. Some of the current extensions are:

- Kernel
- Basic Utilities
- Advanced Utilities
- Administered Systems
- Software Development
- Terminal Interface
- Real Time
- Remote Services

A given System V operating system implementation does not require the extensions; however, when an extension is made part of the implementation, it must conform to the functionality specified by the SVID.

The Base System defines the minimum System V implementation. The Base System itself interfaces to the underlying hardware through the Application Binary Interface (ABI) unique to each processor. The ABI defines:

- Fundamental data types and alignment
- Function calling sequences
- Object files
- Program loading and dynamic linking
- System libraries and macros (such as `ctype.h`)

These will in turn affect such low-level routines as `read()`, `write()`, `open()`, etc., as well as *all* binary data structures. Typically, the software that is affected by an ABI can be:

- Device driver code that interacts with machine-specific hardware
- End-user-level assembly language routines
- Routines that manipulate binary data structures

Of most concern to application programmers are the potential differences in how user-level data structures are represented among applications running Solaris 2.x on the SPARC and x86 processors.

The requirement to exchange data between UNIX systems running on a variety of hardware arose years ago. In order to meet this requirement, the *External Data Representation* (XDR) standard was created. To achieve code exchange to another UNIX System V implementation, you need to specify a format for data that allows this. The definitive method for describing arbitrary data structures in a machine-independent manner is to use the XDR standard. XDR is the fundamental data type of ONC™ (Open Network Computing).

Solaris 2.x provides a complete System V environment encompassing all modules defined in the SVID, as well as those in the X11/NeWS™ windowing system.

SunOS 4.1.x to Solaris 2.x Migration Path

The recommended strategy is to use SunOS 4.1.x as the base for developing SVID-compliant applications. SunOS 4.1.x provides a robust, feature-rich environment that includes the compilers, libraries, and system administration and analysis tools needed to develop applications that can be easily ported to Solaris 2.x. As a first step in the Solaris 2.x porting process, applications still running 4.0.3 should be moved to native SunOS 4.1.x in preparation for the Solaris 2.x porting effort. In most cases, this should be a relatively straightforward, minor port.

The suggested porting strategy is to first move applications away from use of the BSD-based compiler and libraries in SunOS 4.1.x, toward use of the System V-based compiler and libraries in `/usr/5bin` and `/usr/5lib`. These are the SVID-compatible libraries and must be used to produce truly SVID-compatible source. Doing this initial port will position the application to move rapidly to the Solaris 2.x environment, while continuing to support the existing SunOS 4.1.x installed base.

As noted earlier, migrating to SVID compliance in SunOS 4.1.x does not guarantee compliance with the SPARC ABI. However, it is a step along the way towards conformance with the SPARC Compliance Definition version 2.0 (SCD 2.0).

Figure 8-1 illustrates the migration paths that are available when porting SunOS 4.1.x applications to Solaris 2.x.

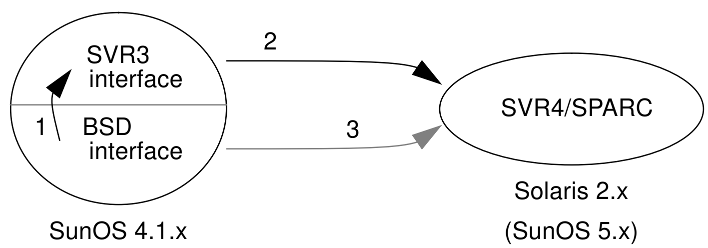

Figure 8-1 Migration Paths to Solaris 2.x

Developers are strongly encouraged to begin using the SVR3 interfaces immediately and to migrate away from the BSD-based portions of SunOS 4.1.x, a process represented by arrow 1 in Figure 8-1. When developers are ready to port to Solaris 2.x, any remaining enhancements or features not already defined in SVR3 can be taken advantage of in the second stage of the porting process, as indicated by arrow 2. Although it is possible to port directly to Solaris 2.x, as represented by arrow 3, that path is *not* recommended.

Compatibility Between SunOS 4.1.x and Solaris 2.x

The source-level interfaces in SunOS 4.1.x and Solaris 2.x are identical in many areas, highly compatible in some areas, and incompatible in others. Figure 8-2 portrays the Venn diagram model that is used in Appendix D of this book to illustrate the basic areas of compatibility and incompatibility between the two operating systems, including the various SunOS commands, system calls, and library routines.

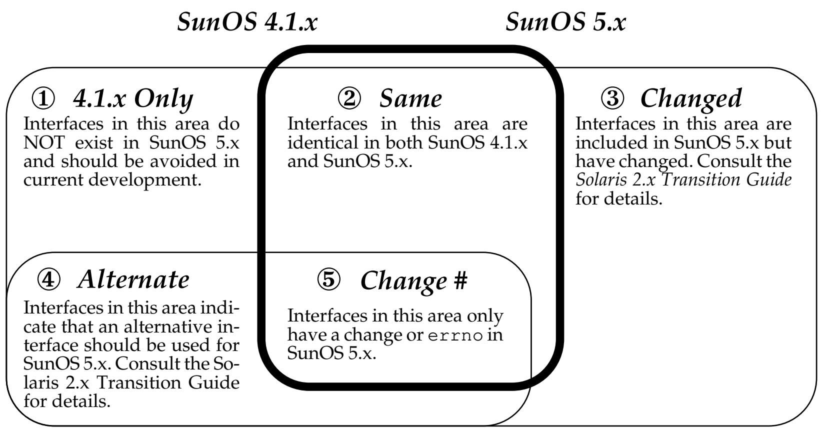

Figure 8-2 System-Call Compatibility

The SVID-compliant interfaces in `/usr/5lib` and `/usr/5bin` are located in area ② of the Venn diagram. "Pure" BSD interfaces that have not been adopted by Solaris 2.x are located in area ①. Commands and system calls that have an alternative interface are located in area ④. Interfaces that remain the same except for their `errno` specifications are located in area ⑤. Interfaces that have changed beyond merely different `errno` specifications are located in area ③.

The System V Migration Tool

Once an application has been brought into 4.1.x, the next step is to ensure that the application uses only the interfaces defined by SVID3. This requires identifying system interfaces used by the application that differ between SunOS 4.1.x and Solaris 2.x and determining whether the differences affect proper execution. Because the process of locating and categorizing every system and library call within an application's source tree can be daunting, SunSoft has developed `svmt`, the System V Migration Tool. To order a copy of `svmt`, please call the SunSoft Catalyst organization.

This program scans a body of source code against a database of all SunOS 4.1.x interfaces and indicates which calls are incompatible with Solaris 2.x and why. The `svmt` is provided to encourage developers to assess the impact of the required porting effort.

The example in Figure 8-3 illustrates the output of `svmt` for a sample program:

```
/* sample.c */
#include <sys/stat.h>
main()
{
    char    *str1="Abcdefghijklmnopqrstuvwxyz";
    char    *str2="abcdeFGHIJKLMNOPQRSTuvwxyz";

    /* routine not in SVR4(strcasecmp) */
    if(strcasecmp(str1, str2) == 0)
    {
        printf("Strings are equal\n");
    }

    /* routine in both SVR4 and SunOS 4.x */
    sleep(3);

    /* routine replaced in SVR4(rindex) */
    printf("partial string of str1 = '%s'\n",
        rindex(str1, 'Q'));
    printf("partial string of str2 = '%s'\n",
        rindex(str2, 'Q'));

    /* routine with new interface in SVR4 */
    if(creat("/tmp/abc", 0644) < 0)
    {
        perror("creat");
    }
}
% svmt sample.c
strcasecmp(3)  sample.c(11)
    In SunOS 4.x.y/sysV:
        This interface is not available in SystemV Release
        3/svidII, however it may be functional under
        4.x.y/sysV.

    In SunOS 5.0:
        This interface is not available in SunOS 5.0.

    In SPARC ABI:
```

Figure 8-3 Output of `svmt` for a Sample Program

```
        This interface is not available in SPARC ABI.

rindex(3)   sample.c(21)
    In SunOS 4.x.y/sysV:
        This interface is not part of SystemV Release
        3/svidII. It is replaced by strrchr(3) in
        4.x.y/sysV.

    In SunOS 5.0:
        This routine is not available in SunOS 5.0. This
routine is replaced by strrchr (3C) in SunOS 5.0.

    In SPARC ABI:
        This routine is not available in SPARC ABI. This
routine is replaced by strrchr (3C) in SPARC ABI.

    Additional Code Reference(s):
        sample.c(23)

creat(2V)   sample.c(26)
    In SunOS 4.x.y/sysV:
        Under SunOS 4.x.y, the mode argument to creat() is
of type
        int, while under 4.x.y/SysV the mode argument is of
type
        mode_t. Also, under 4.x.y/SysV version include
<sys/stst.h>
        while SunOS 4.x.y does not. Also ENOENT is returned
when the
        path points to an empty string.

    In SunOS 5.0:
        Under SunOS 4.x.y, the mode argument to creat() is
of type int, while under SunOS 5.0 the mode argument is of
type mode_t. Also, the SunOS 5.0 version include <fcntl.h>
while SunOS 4.x.y does not. The following symbolic access
modes (<sys/stat.h>) are supported by the SunOS 4.x.y
version of chmod(), but not by the SunOS 5.0 version:
S_IREAD (00400), S_IWRITE (00200), S_IEXEC (00100).
However, the equivalent SunOS 5.0 symbolic access modes,
S_IRUSR (00400), S_IWUSR (00200), S_IXUSR (00100), do have
the same definitions, are defined in SunOS 4.x.y
<sys/stat.h>, and thus should be used. The following errno
flags are valid for the SunOS 4.x.y version of this system
call but are not valid in the SunOS 5.0 version:
```

Figure 8-3 Output of `svmt` *for a Sample Program (Continued)*

```
        EMFILE, ENXIO, EOPNOTSUPP.

    In SPARC ABI:
        Under SunOS 4.x.y, the mode argument to creat() is
of type int, while under SPARC ABI the mode argument is of
type mode_t. Also, the SPARC ABI version include <fcntl.h>
while SunOS 4.x.y does not. The following symbolic access
modes (<sys/stat.h>) are supported by the SunOS 4.x.y
version of chmod(), but not by the SPARC ABI version:
S_IREAD (00400), S_IWRITE (00200), S_IEXEC (00100).
However, the equivalent SPARC ABI symbolic access modes,
S_IRUSR (00400), S_IWUSR (00200), S_IXUSR (00100), do have
        the same definitions, are defined in SunOS 4.x.y
<sys/stat.h>, and thus should be used. The following errno
flags are valid for the SunOS 4.x.y version of this system
call but are not valid in the SPARC ABI version:
        ENXIO, EOPNOTSUPP.
```

Figure 8-3 Output of `svmt` *for a Sample Program (Continued)*

Solaris 2.x Source Compatibility

Appendix D ("SunOS 4.1.x—SunOS 5.x Compatibility Diagrams") uses Venn diagrams to illustrate these differences. In addition, this appendix has detailed information, including a set of migration reference tables that indicate each system call or library routine interface and migration reference notes that describe any changes between the two releases.

For example, an easily illustrated interface that changes between SunOS 4.1.x, and 5.x is `send()`. The change is simple: one error condition is no longer supported — `ENOBUFS`. For most applications, this is unlikely to have an impact on proper operation. For those applications that do check for an `ENOBUFS` error condition, some minor modifications will be necessary.

Another example is found in the `chmod()` system call. The migration reference notes reveal that the changes are in two areas. First, the error condition `EINVAL` is not supported in Solaris 2.x. This can be accommodated as discussed above. Second, the symbolic access modes `S_IREAD`, `S_IWRITE`, and `S_IEXEC` are present in SunOS 4.1.x, but are not supported in Solaris 2.x. As it turns out, these modes are equivalent to the SVR4 modes `S_IRUSR`, `S_IWUSR`, and `S_IXUSR`. Furthermore, these modes have the same definitions in SunOS 4.1.x. Therefore, if the incompatible modes have been used, they should be changed to the compatible modes to achieve the goal of source compatibility between 4.1.x and Solaris 2.x.

With few exceptions, the SunOS 4.1.x libraries conform to the System V Interface Definition. Generally, these exceptions occur when system or library calls have matching names but incompatible effects, return values, error conditions, or parameter usages in SunOS 4.1.x and the SVID.

It is also important to examine other ways that an application may interface with UNIX in a system-dependent way. An application may depend on certain system files being present or using a particular format. These system files and formats may change between SunOS 4.1.x and SunOS 5.x; such files and formats should not be hard-coded into the SunOS 4.1.x version of an application.

For example, directory files in SunOS 4.1.x do not have the same format as directory files in other System V file system implementations. Programs that read special files such as directories, must be converted to use the appropriate interface for that file format. For directories, interfaces include `opendir()`, `readdir()`, `telldir()`, `seekdir()`, `rewinddir()`, and `closedir()`. The new include file, `<dirent.h>`, should be used, which replaces the old header file, `<sys/dir.h>`, used in BSD 4.2 and 4.3 and in previous releases of SunOS. Furthermore, with the use of this new file, the `readdir()` routine returns directory entries whose structure is named `struct dirent` rather than `struct direct`. Programs that already implement the BSD directory interfaces should be modified to conform to the SVID. For further information, refer to the man page for `directory`.

Other examples of required changes are programs that deal with asynchronous communication ports, pseudo-terminals, and the special interface `/dev/tty`. The programs must be converted to use the `termio/termios` structure rather than the `sgttyb` structure. Currently, under SunOS 4.1.x, `<sgtty.h>` is supported as a STREAMS module, called `ttcompat`, that translates the `ioctl` calls supported by `<sgtty.h>` into `ioctl` calls supported by the `termio(4)` interface. For further information, please refer to the man pages for `termio`, `ttcompat`, and `tty`.

BSD Compatibility

Some BSD functionality not included in Solaris 2.x can be accessed in SunOS 5.x through the BSD Compatibility Package. This package contains BSD commands, library routines, header files, and system calls that are not part of Solaris 2.x. Use of this package is highly discouraged, however, for it decreases application performance.

At some later date, the interfaces in the Compatibility Package will no longer be supported. They are only included in SunOS 5.x to simplify the initial transition to compliance with SVID3.

If the Compatibility Package is installed on your system, header files and libraries called by the C compiler and linker are located in `/usr/ucbinclude` and `/usr/ucblib`. These differences should be noted; the proper flags must be set in the environment via the `PATH` environment variable and in the makefile.

By setting the `PATH` variable so `/usr/ucb` comes before the default path directory `/usr/bin`, `/usr/ucb/cc` will be used to compile C programs. `/usr/ucb/cc` sets its default paths to pick up, in this order:

1. User-specified `include` directories and libraries
2. Compatibility `include` files and directories
3. Default System V headers and libraries, if unresolved symbols remain

In addition to library locations, some differences in available header files and header file definitions exist between Solaris 2.x and the BSD Compatibility Package. In some cases, certain BSD header files exist only in the Compatibility Package. Also, some header files found in both BSD and Solaris 2.x systems may differ. If the recommended strategy is followed and applications are SVID3-compliant, this should have very little impact, if any, on the application.

BSD header files not included in Solaris 2.x include:

- `dbm.h`
- `fp.h`
- `mp.h`
- `ndbm.h`
- `strings.h`
- `struct.h`
- `sunfp.h`
- `sysexits.h`
- `sys/ieeefp.h`

Header files that differ between Solaris 2.x and the BSD Compatibility Package include:

- `assert.h`
- `regexp.h`
- `setjmp.h`
- `signal.h`
- `stdio.h`
- `unistd.h`
- `sys/dirent.h`
- `sys/fcntl.h`
- `sys/file.h`
- `sys/param.h`
- `sys/resource.h`
- `sys/signal.h`
- `sys/types.h`
- `sys/vfs.h`
- `sys/wait.h`

BSD system calls residing in /usr/ucblib/libucb.a include:

System Call	Action
getdtablesize	calls getrlimit() system call
gethostid	calls sysname() system call
gethostname	calls sysname() system call
getrusage	calls times() system call
gettimeofday	calls hrtcntl(); note that this version of gettimeofday() does not use the second argument
killpg	calls kill() system call
setreuid	calls setuid() and seteuid() system calls
setregid	calls setgid() and setegid() system calls
settimeofday	calls stime() system call; note that the second argument is ignored
syscall	calls trap mechanism
utimes	calls utime() system call
wait3	calls waitid() system call

BSD-style signal handling routines in /usr/ucblib/libucb.a include:

Routine	Action
setjmp, longjmp, _setjmp, _longjmp	calls sigsetjmp(),siglongjmp()
psignal, _psignal	calls psignal()
sigblock,sigsetmask	calls sigprocmask()
sigpause	calls sigsuspend()
siginterrupt, signal	calls version of sigvec()
sigvec	calls sigaction()

Routines located in /usr/ucblib/libucb.a that differ slightly from Solaris 2.x versions include:

- fopen
- fprintf
- nice
- nlist
- printf
- rand
- sleep
- times
- vfprintf
- vprintf
- vsprintf

Routines in `/usr/ucblib/libmp.a` that perform arithmetic on integers of arbitrary length include:

- `gcd`
- `itom`
- `madd`
- `mcmp`
- `mdiv`
- `mfree`
- `min`
- `mout`
- `msqrt`
- `msub`
- `mtox`
- `mult`
- `pow`
- `rpow`
- `sdiv`

Routines in `/usr/ucb/libdbm.a` that maintain key/content pairs in a database using database routines include:

- `dbmclose`
- `dbminit`
- `delete`
- `fetch`
- `firstkey`
- `nextkey`
- `store`

Other routines in /usr/ucblib/libucb.a include:

- alloca
- alphasort
- bcmp
- bcopy
- bzero
- copysign
- dbm_clearerr
- dbm_close
- dbm_delete
- dbm_error
- dbm_fetch
- dbm_firstkey
- dbm_nextkey
- dbm_open
- decimal_to_double
- decimal_to_extended
- decimal_to_single
- double_to_decimal
- econvert
- extended_to_decimal
- fconvert
- fp_class
- fsetstate
- ftime
- gconvert
- getwd
- ieee_handler
- index
- initstate
- isnan
- mkstemp
- random
- re_comp
- re_exec
- rindex
- scalbn
- scandir
- seconvert
- setbuffer
- setlinebuf
- sfconvert
- sgconvert
- single_to_decimal
- srandom
- timezone
- ualarm
- usleep

SVID3 vs. SCD 2.0

SunOS 4.1.x provides tools for the development of applications that are source-level-compatible with the standard Solaris 2.x operating system. However, it is important to note the difference between SVID compatibility and compliance with the SPARC ABI. Although applications can be developed on SunOS 4.1.x to conform with the SVID, there is no facility to test SunOS 4.1.x applications against the Solaris 2.x SPARC Application Binary Interface as defined in the SCP.

SVID3 applies to all system architectures. It defines what is included in Solaris 2.x at the Application Programming Interface (API) level, including interfaces such as system calls and standard library routines. Although the ABI does have portions that apply to all architectures, it also includes portions that are specific to a particular architecture (SPARC). Along with the source-level application source interface, the ABI defines architecture-specific details, such as data representations, stack frames, trap interfaces, object file formats, and certain aspects of the dynamic linker at the binary level.

Bibliography

UNIX System V Interface Definition, 3rd Edition, Volumes 1-5, Addison-Wesley Publishing Company, Reading, MA.

AT&T UNIX System V Release 4 Programmer's Reference Manual, Prentice Hall, Englewood Cliffs, NJ.

Migration Guide for System V Developers, UNIX International.

Solaris 2.1 Migration Guide, Part Number 801-2815-10, SunSoft, Inc., Mountain View, CA.

SunOS 4.1 Reference Manual, Part Number 800-3827-10, SunSoft, Inc., Mountain View, CA.

SunOS 5.1 Writing Device Drivers, Part Number 801-2871-10, SunSoft, Inc., Mountain View, CA.