Advanced Programming in the UNIX Environment — *UNIX Standardization and Implementations*

Hop Lee
hoplee@bupt.edu.cn

School of Information and Communication Engineering

Table of Contents I

Table of Contents II

UNIX Standardization

- ISO C
- POSIX.1
- XPG3

ISO C I

- ANSI Standard X3.159-1989 for the C programming language was approved in late 1989.
- This standard was also adopted as International Standard ISO/IEC[1] 9899:1990.
- ANSI[2] is the U.S. member in the ISO[3].
- The C standard is now maintained and developed by the ISO/IEC international standardization working group for the C programming language, known as ISO/IEC JTC1/SC22/WG14, or WG14 for short.
- The intent of the ISO C standard is to provide portability of conforming C programs to a wide variety of operating systems, not only the UNIX System.

ISO C II

- This standard defines not only the syntax and semantics of the programming language but also a standard library.
- In 1999, the ISO C standard was updated and approved as ISO/IEC 9899:1999, largely to improve support for applications that perform numerical processing.
- This standard defines not only the syntax and semantics of the programming language but also a standard library.
- The ISO C library can be divided into 24 areas, based on the headers defined by the standard (see Figure 2.1).

[1] International Electrotechnical Commission
[2] American National Standards Institute
[3] International Organization for Standardization

IEEE POSIX

- POSIX is a family of standards developed by the IEEE.
- POSIX stands for *Portable Operating System Interface*. Include 1003.1:the operating system interface, 1003.2: shell and utilities, 1003.7: system administration, etc.
- The newest version of POSIX should be IEEE Std. 1003.1-2008, or ISO/IEC 9945:200.

The Single UNIX Specfication

- The Single UNIX Specification (SUS), a superset of the POSIX.1 standard, specifies additional interfaces that extend the functionality provided by the POSIX.1. POSIX.1 is equivalent to the Base Specifications portion of the SUS.
- The *X/Open System Interfaces* (XSI) option in POSIX.1 describes optional interfaces and defines which optional portions of POSIX.1 must be supported for an implementation to be deemed XSI conforming.
- The Single UNIX Specification is a publication of The Open Group, which was formed in 1996 as a merger of X/Open and the Open Software Foundation (OSF), both industry consortia.
- The newest version of SUS is SUS 2010 V4. We'll refer to this as SUSv4.

FIPS

- FIPS stands for *Federal Information Processing Standard*, and these standards are published by the U.S. Government.
- FIPS 151-1(April 1989) is based on the IEEE Std. 1003.1-1988 and a draft of the ANSI C standard.
- FIPS 151-2 requires some features that POSIX.1 lists as optional.
- The POSIX.1 FIPS has since been withdrawn, so we won't consider it further in this lecture.

UNIX System Implementations

- The previous section described three standards are interface specifications. These standards are taken by vendors and turned into actual implementations.
- The ancestor of all implementations are Sixth Edition(1976) and Seventh Edition(1979) of the UNIX Time-Sharing System on PDP-11.
- Three branches of the family tree evolved:
  - AT&T led to System III and System V(commercial versions);
  - BSD led to the 4.xBSD implementations;
  - AT&T Bell Laboratories led to the 8th, 9th, and 10th Editions(ended in 1990).

System V Release 4

- SVR4 is a product of AT&T UNIX System Laboratories (USL).
- The SVR4 source code was released in late 1989, with the first end-user copies becoming available during 1990.
- SVR4 conforms to both the POSIX 1003.1 standard and the XPG3 standard.
- AT&T also publishes the *System V Interface Definition (SVID)*. Issue 3 of the *SVID* specified the functionality that an operating system must offer to qualify as a conforming implementation of UNIX SVR4.

4.4BSD

- The BSDs are produced and distributed by the *Computer Systems Research Group* (CSRG) at the *University of California* at Berkeley.
- Throughout this text we use the term `4.4BSD` to refer to the BSD system being described.

FreeBSD

- FreeBSD is based on the 4.4BSD-Lite.
- The FreeBSD project was formed to carry on the BSD line after the CSCG decided to end its work on the BSD versions of the UNIX operating system, and the 386BSD project seemed to be neglected for too long.
- All software produced by the FreeBSD project is freely available in both binary and source forms.
- The FreeBSD 8.0 operating system was one of the four operating systems used to test the examples in this book.

Linux

- Linux is an operating system that provides a rich programming environment similar to that of a UNIX System; it is freely available under the GNU Public License.
- Linux was created in 1991 by Linus Torvalds as a replacement for MINIX. Many developers across the world volunteered their time to use and enhance it.
- The Ubuntu 12.04 was one of the operating systems used to test the examples in this book. That distribution uses the Linux kernel v3.2.0.

Max OS X

- Mac OS X is based on entirely different technology than prior versions. A not too old core version is called “Darwin,” and is based on a combination of the Mach kernel, the FreeBSD, and an object-oriented framework for drivers and other kernel extensions.
- As of version 10.5, the Intel port of Mac OS X has been certified to be a UNIX system.
- Mac OS X version 10.6.8 (Darwin 10.8.0) was used as one of the operating systems to test the examples in this book.

Solaris

- Solaris is the version of the UNIX System developed by Sun Microsystems (now Oracle).
- Solaris is based on System V Release 4, but includes more than fifteen years of enhancements from the engineers at Sun Microsystems.
- In 2005, Sun Microsystems released most of the Solaris operating system source code to the public as part of the OpenSolaris open source operating system.
- The Solaris 10 UNIX system was one of the operating systems used to test the examples in this book.

Relationship of Standards and Implementations

- The standards that we've mentioned defined a subset of any actual system.
- The focus of this book is to describe four real UNIX systems: FreeBSD 8.0, Linux 3.2.0, Mac OS X 10.6.8, and Solaris 10.

Limits I

- There are many magic numbers and constants that are defined by the implementations.
- Two types of limits are needed:
  - compile-time limits
  - run-time limits
- Compile-time limits can be defined in headers that any program can include at compile time. But run-time limits require the proecess to call a function to obtain the value of the limit.
- Additinally, some limits can be fixed on a given implementation yet vary on another implementation.
- To solve the conflicts, three types of limits are provided:
  - compile-time limits

Limits II

- run-time limits not associated with a file or directory
- run-time limits that are associated with a file or directory

ISO C Limits

- All of the compile-time limits defined by ISO C are defined in the file `limits.h`.

POSIX Limits I

- POSIX.1 defines numerous constants that deal with implementation limits of the operating system.
- There are 33 different limits and constants. These are divided into the following seven categories:
  1. Numerical limits: `LONG_BIT`, `SSIZE_MAX`, and `WORD_BIT`
  2. Minimum values: the 25 constants in Figure 2.8
  3. Maximum value: `_POSIX_CLOCKRES_MIN`
  4. Run-time increasable value: `CHARCLASS_NAME_MAX`, `COLL_WEIGHTS_MAX`, `LINE_MAX`, `NGROUPS_MAX`, and `RE_DUP_MAX`
  5. Run-time invariant values (possible indeterminate):the 17 constants in Figure 2.9 (plus an additional four constants introduced in Section 12.2 and three constants introduced in Section 14.5)

POSIX Limits II

6. Other invariant values: `NL_ARGMAX`, `NL_MSGMAX`, `NL_SETMAX`, and `NL_TEXTMAX`
7. Pathname variable values: `FILESIZEBITS`, `LINK_MAX`, `MAX_CANON`, `MAX_INPUT`, `NAME_MAX`, `PATH_MAX`, and `PIPE_BUF`, and `SYMLINK_MAX`

- Check `/usr/include/x86_64-linux-gnu/bits/local_lim.h`, `/usr/include/x86_64-linux-gnu/bits/posix1_lim.h` and `/usr/include/x86_64-linux-gnu/bits/posix2_lim.h` for the various limits and their values under Mac OS X 10.10.2.

XSI Limits

- The XSI option also defines constants representing implementation limits. They include:
  1. Minimum values: the five constants in Figure 2.10
  2. Runtime invariant values, possibly indeterminate: `IOV_MAX` and `PAGE_SIZE`

sysconf, pathconf, and fpathconf Functions I

- We've listed various minimum values that an implementation must support, but how do we find out the limits that a particular system actually supports?
- The run-time limits are obtained by calling one of the following three functions:

```
1 #include <unistd.h>
2 long sysconf(int name);
3 long pathconf(const char *pathname, int name);
4 long fpathconf(int fd, int name);
```

- The parameters used by these three functions are restricted:
  - Constants beginning with _SC_ are used by sysconf
  - Constants beginning with _PC_ are used by either pathconf or fpathconf

sysconf, pathconf, and fpathconf Functions II

- There are some restrictions for the *pathname* parameter to pathconf and the *fd* parameter to fpathconf. If any of those restriction aren't met, the results are undefined.
- Example (Figure 2.13, standards/makeconf.awk). This awk(1) program builds a C program that prints the value of each pathconf and sysconf symbol.
- The awk program reads two input files—pathconf.sym and sysconf.sym—that contain lists of the limit name and symbol, separated by tabs. All symbols are not defined on every platform, so the awk program surrounds each call to pathconf and sysconf with the necessary #ifdef statements.
- Example (Figure 2.14, standards/conf.c), generated by the awk program, prints all these limits, handling the case in which a limit is not defined.

sysconf, pathconf, and fpathconf Functions III

- Figure 2.15 on textbook summarizes the results from Figure 2.14 for the four systems we discuss in this lecture.

Indeterminate Runtime Limits

- We mentioned that some of the limits can be indeterminate. The problem we encounter is that if these limits aren't defined in the `limits.h` header, we can't use them at compile time. But they might not be defined at runtime if their value is indeterminate!
- Let's look at two specific cases: allocating storage for a pathname and determining the number of file descriptors.
- Pathnames. Example (Figure 2.16, `lib/pathalloc.c`).
- Maximum Number of Open Files. Example (Figure 2.17, `lib/openmax.c`).

Options I

- We saw the list of POSIX.1 options in Figure 2.5 and discussed XSI option groups in Section 2.2.3. If we are to write portable applications that depend on any of these optionally supported features, we need a portable way to determine whether an implementation supports a given option.
- For each option, we have three possibilities for a platform's support status.
  1. If the symbolic constant is either undefined or defined to have the value -1, then the corresponding option is unsupported by the platform at compile time. It is possible to run an old application on a newer system where the option is supported, so a runtime check might indicate the option is supported even though the option wasn't supported at the time the application was compiled.

Options II

2. If the symbolic constant is defined to be greater than zero, then the corresponding option is supported.
3. If the symbolic constant is defined to be equal to zero, then we must call sysconf, pathconf, or fpathconf to determine whether the option is supported.

Feature Test Macros

- The headers define numerous POSIX.1 and XSI symbols, as we've described. Even so, most implementations can add their own definitions to these headers, in addition to the POSIX.1 and XSI definitions. If we want to compile a program so that it depends only on the POSIX definitions and doesn't conflict with any implementation-defined constants, we need to define the constant `_POSIX_C_SOURCE`. All the POSIX.1 headers use this constant to exclude any implementation-defined definitions when `_POSIX_C_SOURCE` is defined.
- The constants `_POSIX_C_SOURCE` and `_XOPEN_SOURCE` are called **feature test macros**.

Primitive System Data Types

- Historically, certain C data types have been associated with certain UNIX system variables.
- The header `sys/types.h` defines some implementation-dependent data types with the C `typedef` facility, called the **primitive system data types**.
- Figure 2.21 lists many of the primitive system data types that we'll encounter in this text.

Differences Between Standards

- All in all, these various standards fit together nicely. Our main concern is any differences between the ISO C standard and POSIX.1, since the Base Specifications of the Single UNIX Specification and POSIX.1 are one and the same. Conflicts are unintended, but if they should arise, POSIX.1 defers to the ISO C standard. However, there are some differences.

Summary

- Much has happened with the standardization of the UNIX programming environment over the past two and a half decades. We've described the dominant standards—ISO C, POSIX, and the Single UNIX Specification—and their effect on the four platforms that we'll examine in this text. These standards try to define certain parameters that can change with each implementation, but we've seen that these limits are imperfect. We'll encounter many of these limits and magic constants as we proceed through the text.
- The bibliography specifies how to obtain copies of the standards discussed in this chapter.

The End

The End of Chapter 2.