APE — The ANSI/POSIX Environment

Howard Trickey

Introduction

When a large or frequently-updated program must be imported or exported to and from Plan 9, the ANSI/POSIX environment known as APE can be useful. APE combines the set of headers and object code libraries specified by the ANSI C standard (ANSI X3.159-1989) with the POSIX operating system interface standard (IEEE 1003.1-1990, ISO 9945-1). Using APE will cause slower compilation and marginally slower execution speeds, so if the importing or exporting happens only infrequently, due consideration should be given to using the usual Plan 9 compilation environment instead. Another factor to consider is that the Plan 9 header organization is much simpler to remember and use.

There are some aspectes of required POSIX behavior that are impossible or very hard to simulate in Plan 9. They are described below. Experience has shown, however, that the simulation is adequate for the vast majority of programs. The much more common problem is that many programs use functions or headers not defined by POSIX. APE has some extensions to POSIX to help in this regard.

Pcc

The `pcc` command acts as a front end to the Plan 9 C compilers and loaders. It runs an ANSI C preprecessor over source files, using the APE headers to satisfy `#include` <*file*> directives; then it runs a Plan 9 C compiler; finally, it may load with APE libraries to produce an executable program. The document *How to Use the Plan 9 C Compiler* explains how environment variables are used by convention to handle compilation for differing architectures. The environment variable `$objtype` controls which Plan 9 compiler and loader are used by `pcc`, as well as the location of header and library files. For example, if `$objtype` is `mips`, then `pcc` has `cpp` look for headers in `/mips/include/ape` followed by `/sys/include/ape`; then `pcc` uses `vc` to create `.v` object files; finally, `vl` is used to create an executable using libraries in `/mips/lib/ape`.

Symbols

The C and POSIX standards require that certain symbols be defined in headers. They also require that certain other classes of symbols not be defined in the headers, and specify certain other symbols that may be defined in headers at the discretion of the implementation. POSIX defines *feature test macros*, which are preprocessor symbols beginning with an underscore and then a capital letter; if the program `#defines` a feature test macro before the inclusion of any headers, then it is requesting that certain symbols be visible in the headers. The most important feature test macro is `_POSIX_SOURCE`: when it is defined, exactly the symbols required by POSIX are visible in the appropriate headers. Consider `<signal.h>` for example: ANSI defines some names that must be defined in `<signal.h>`, but POSIX defines others, such as `sigset_t`, which are not allowed according to ANSI. The solution is to make the additional symbols visible only when `_POSIX_SOURCE` is defined.

When you export a program, it helps to know whether the program fits in one of the following categories:

1. Strictly conforming ANSI C program. It only uses features of the language, libraries, and headers explicitly required by the C standard. It does not depend on unspecified, undefined, or implementation-dependent behavior, and does not exceed any minimum implementation limit.
2. Strictly conforming POSIX program. Similar, but for the POSIX standard.

3. Some superset of POSIX, with extensions. Each extension is selected by a feature test macro, so it is clear which extensions are being used.

With APE, if headers are always included to declare any library functions used, then the set of feature test macros defined by a program will show which of the above categories the program is in. To accomplish this, no symbol is defined in a header if it not required by the C or POSIX standard, and those required by the POSIX standard are protected by #ifdef _POSIX_SOURCE. For example, <errno.h> defines EDOM, ERANGE, and errno, as required by the C standard. The C standard allows more names beginning with E, but our header defines only those unless _POSIX_SOURCE is defined, in which case the 35 more symbols required by POSIX are defined. This means that a program which uses ENAMETOOLONG cannot masquerade as a strictly conforming ANSI C program.

Pcc does not predefine any preprocessor symbols except those required by the ANSI C standard: __STDC__, __LINE__, __FILE__, __DATE__, and __TIME__. Any others must be defined in the program itself or using -D on the command line.

Extensions

The pedagogy enforced by only putting required names in the headers is useful for exporting programs, but it gets in the way when importing programs. The compromise is to allow additional symbols in headers, additional headers, and additional library functions, but only under control of extension feature test macros. The following extensions are provided; unless otherwise specified, the additional library functions are in the default APE library.

- _LIBG_EXTENSION. This allows the use of the Plan 9 graphics library. The functions are as described in the Plan 9 manual (see *graphics*(2)) except that div had to be renamed ptdiv. Include the <libg.h> header to declare the needed types and functions, and use -lg on the pcc command line to load the graphics functions.
- _LIMITS_EXTENSION. POSIX does not require that names such as PATH_MAX and OPEN_MAX be defined in <limits.h>, but many programs assume they are defined there. If _LIMITS_EXTENSION is defined, those names will all be defined when <limits.h> is included.
- _BSDTIME_EXTENSION. The header <sys/time.h> is not defined by POSIX. This extension allows the inclusion of that header, with its definitions of the timeval and timezone structures.
- _SELECT_EXTENSION. This extension allows the inclusion of the <select.h> header, which defines the Berkeley select function and associated types and macros for dealing with multiple input sources. The <sys/time.h> header must be included before <select.h>, so the _BSDTIME_EXTENSION is needed too.
- _PCONS_EXTENSION. This defines a ‘‘pseudo-tty’’ capability. If _PCONS_EXTENSION is defined, <pcons.h> may be included; it defines the functions pcons(void) and pushtty(void). Pcons arranges for a Plan 9 file server to provide the /dev/cons and /dev/consctl files. It also provides /dev/mcons, which is the ‘‘master end’’ of the pseudo-tty: characters written to /dev/mcons can be read from /dev/cons (or /dev/tty, a synonym) and vice versa. POSIX tty line processing is supported: the contents of a termios structure can be written to or read from the /dev/pttyctl file in the format

  ```
  "stty %8ux %8ux %8ux %8ux "
  ```

  (the numbers are the c_iflag, c_oflag, c_cflag, and c_lflag), followed by NCCS bytes of c_cc data. The pushtty function uses pcons to arrange that the standard input, output, and error files behave as ttys are supposed to according to POSIX. (This should not have to be done explicitly, but the cost of supporting tcgetattr and tcputattr is sufficiently high that it seemed better to not make it the default.)
- _NET_EXTENSION. This extension allows inclusion of <libnet.h>, which defines the networking functions described in the Plan 9 manual page *dial*(2). Programs using them must be loaded with -lnet.
- _REGEXP_EXTENSION. This extension allows inclusion of <libregexp.h>, which defines the

regular expression matching functions described in the Plan 9 manual page *regexp*(2X). Programs using them must be loaded with `-lregexp`.

- `_RESEARCH_SOURCE`. This extension enables a small library of functions from the Tenth Edition Unix Research System (V10). These functions and the types needed to use them are all defined in the `<libv.h>` header, and programs using them must be loaded with `-lv`. The provided functions are: `popen` and `pclose` (common extensions to ANSI C stdio; the `<stdio.h>` header must have been included); `telldir` and `seekdir` (common extensions to the ANSI C `readdir` suite; `<dirent.h>` must have been included); `lstat`, `symlink`, and `readlink` (dummy functions for Plan 9, since there are no symbolic links); `srand`, `rand`, `nrand`, `lrand`, and `frand` (better random number generators); `getopt` (program argument handler); `getpass`, `tty_echoon`, `tty_echooff` (for dealing with the common needs for mucking with terminal characteristics); `min` and `max`; `nap`; `setfields`, `getfields`, and `getmfields` (for parsing a line into fields); and `ftw` (for walking a file tree). See the Research Unix Programmer's Manual for a description of these functions.

Common Problems

Some large systems, including X11, have been ported successfully to Plan 9 using APE. The problems encountered fall into three categories: (1) non- ANSI C/POSIX features used; (2) inadequate simulation of POSIX functions; and (3) compiler/loader bugs. By far the majority of problems are in the first category.

POSIX is just starting to be a target for programmers. Most existing code is written to work with one or both of a BSD or a System V Unix. System V is fairly close to POSIX, but there are some differences. Also, many System V systems have imported some BSD features that are not part of POSIX. Here are some common non-POSIX things used in current programs, and what to do about them.

- `sys_errlist` and `sys_nerr` globals. Use `strerror` instead.
- Third (environment) argument to `main`. Use the `environ` global instead.
- `OPEN_MAX`, `PATH_MAX`, etc., assumed in `<limits.h>`. Rewrite to call `sysconf` or define `_LIMITS_EXTENSION`.
- `isascii` assumed in `<ctype.h>`. Let the dead codeset go: leave the test out, or better still, convert to use multibyte characters (c.f. runes in Plan 9). Or define your own `isascii` macro.
- `<strings.h>`, `index`, and `rindex`. Use `<string.h>`, `strchr`, and `strrchr` instead.
- `mktemp`. Rewrite to use ANSI C `tempnam` or `tmpfile`.
- `<varargs.h>`. Rewrite to use `<stdarg.h>`.
- `popen` assumed in `<stdio.h>`. Use `_RESEARCH_SOURCE` extension, include `<libv.h>`, and load with `-lv`. Similarly for `telldir`, `seekdir`, `lstat`, `srand`, `rand`, `nrand`, `lrand`, `frand`, and `getopt`.
- `select`. Define `_BSDTIME_EXTENSION` and `_SELECT_EXTENSION` and include `<sys/times.h>` and `<select.h>`.
- Socket or stream stuff. Rewrite to use the Plan 9 networking functions, define `_NET_EXTENSION`, include `<libnet.h>` and load with `-lnet`.

The second class of problems has to do with inadequacies in the Plan 9 simulation of POSIX functions. As long as this list is, these shortcomings have rarely gotten in the way (except, perhaps for the `link` problem).

- Functions for setting the userid, groupid, effective userid and effective groupid do not do anything useful. The concept is impossible to simulate in Plan 9. `Chown` also does nothing.
- `execlp` and the related functions do not look at the `PATH` environment variable. They just try the current directory and `/bin` if the pathname is not absolute.
- Advisory locking via `fcntl` is not implemented.
- `isatty` is hard to do correctly. The approximation used is to see if the file is the same as

`/dev/cons`. Sometimes this cannot be determined, and sometimes this is not the right test.

- With `kill`, the behavior mandated for negative arguments (killing process groups) is not implemented.
- `link` always fails.
- With `open`, the `O_NOCTTY` option has no effect. The concept of a ‘‘controlling’’ tty is foreign to Plan 9.
- `setsid` does a `forkpgrp`, which is only approximately the right behavior.
- The functions dealing with stacking signals, `sigpending`, `sigprocmask` and `sigsuspend`, do not work.
- `system` uses `rc` instead of `sh` to interpret the argument. Use of `system` is discouraged for security reasons anyway.
- `tcgetattr` and `tcsetattr` only work if `pcons` or `pushtty` has been executed (see the discussion about the `_PCONS_EXTENSION`, above).
- `umask` has no effect, as there is no such concept in Plan 9.
- `utime` has no effect, because the time can’t be changed in Plan 9.