Portable Programming

Orna Agmon

ladypine@vipe.technion.ac.il

Haifux 26.5.2003

TOC

I built it for Linux, I want people to use Linux to use my application.
After all, Linux is the best platform!

Why Should I Write Portable (Cross-Platform) Code

- Be able to compile and test on various platforms.
- Latent bugs may be more obvious on other platforms. Cross platform testing improves the code quality.
- Make use of platform-specific tools, such as valgrind (on Linux) or third (on OSF1)
- Be able to use whatever machine falls into your hands for the purpose of running your code.
- Be able to use multi-architechtural PVM.

Essence of Portability

Using standards :

- Verifying (by documentation) the standard behavior of tools before using them. i.e., what the tools are supposed to do, and not what they usually do.
- Relying only on standard behavior of functions and tools. Avoiding possibly cool system-special features.
- Using standards in a level which is as low as possible. (e.g., in cross-platform source rather than a widely portable compiler or linker)
- Wrapping tools in order to guaranty at least that they do not crash when accepting an undesired syntax.

Essence of Portability

Using portable tools (languages, compilers, OSes) and features:

- Widely available
- Free software

Actually laying hands: on other systems, compilers, OSes, shells, architectures, and building and running on them.

Documenting in free and open formats (pdf, ps...).

Level of Required/ Desired Portability

- Many unknown systems or just two known ones? What are expected future developments?
- Is the software required to work on other platforms, or also have its look'n'feel?
- How system-specific is the code?

The scope of this lecture will be user space programs, and where to expect systems to vary. Some hints may even be given as to how to solve these issues.

Unix- Windows Solutions

- Emulationof UNIX environment: cygwin , Packages of Gnu tools for Windows.
- Emulation of Windows environment: wine
- Scripting: perl, python, tcl/tk
- Common user interface libraries
- Specific code (depending on the type of machine, discovered by configure)

C standards

- ISO C (**ANSI C**) since 1989. Widely supported, hence preferred standard.
- K&R (the first book on C by Kernighan and Ritchie) C. Old Standard.
- C9X. Not widely supported by compilers.

C standards

- gcc is highly portable itself, but some extensions are not. In gcc, the "-pedantic" option will warn about the use of gcc extensions.
- macros from <features.h> will enforce usage of chosen standards in the source level instead of compiler level.
- Example: **for (int i=0;** $i < 10$**; ++i){}**. i may be declared just within the loop (gcc, compiled as c), stay declared after the loop (Visual C, compiled as cpp), or the syntax may not compile at all (cc of OSF1, compiled as c).

"The good things about standards is that there are so many different ones to choose from."

—Patty Seybold, 1988.

Data Types

- Some C types specify minimal sizes only: int is at least 16 bit, long is at least 32 bit. short is exactly 16 bit, char is exactly 8.
- Exact data types are defined in C99 in <stdint.h>. On openBSD, for example, the same types may be defined in <sys/types.h>. Solaris: <sys/int_types.h>. The purpose of autoheader is to solve these issues.
- <limits.h> supplies, well, limits! (maximal values that data types can hold)
- Many Fortran compilers have parameters to define the sizes of variables, such as “ -i4 -r8”. Those parameters enforce consistency within a program.

Data Types still...

- Usage of special types will ensure consistency with system libraries: **size_t** for sizes, **time_t** for the return value of the **time** function. Here, again, there may be differences regarding location of definitions, and autoheader comes to assist.
- Consistency within the software can be achieved by using typedefs, defined in one header for the whole package. Those types must be consistent for each system, though they may vary between systems.
- Bit operations are dangerous regarding varying data sizes.

Casting pointers to Data Types

Casting pointers is dangerous. Period.

- For example, the space of one 8-byte real will accommodate four 2-byte int values on some system, and two 4-byte int values on others) .
- Casting to larger data types, for example (int *) to (float *), may lead to illegal addresses.
- It is not even necessary for sizeof(int)*2=sizeof(double).

Other Stuff

- Usage of static memory (relevant to FORTRAN)
- Linking languages: the name of the compiled function varies. Some compilers may add another _ in the end of subroutines which have _ within their name. Example: when the FORTRAN subroutine uses myfunc and my_func (both in C), the linker will look for a C subroutines called myfunc_ and my_func__ .
- Stack size
- memory size.

Endianness

Endianness is architecture dependent.

- **Little-endian**: the least significant byte is at the lowest address in memory. Examples: Intel, Alpha.
- **Big-endian**: the most significant byte is at the lowest address in memory. Examples: MIPS, Sparc.

Relevant when storing binary data or communicating it.

Endianness- Solutions

- Avoid storing binary data: store ASCII. Free bonus- easily read by a human.
- Communicate using wrappers: PVM
- Another reason to avoid casting of pointers. Avoid endianness "games": For example, do not do in C:
  ```
  int i = 4;
  char c = *(char *) i;
  ```
  Do not do in FORTRAN an equivalence between character and integer.
- Use conversion functions between "Network Endianness" (big endianness, used in TCP/IP protocols) and "Host endianness"- (htons, ntohs, htonl, ntohl). The functions depend on the lengths of **int** and **long**.

C Structure Layout

Compiler dependent. In FORTRAN- common layout.

- Variables are always in the same order.
- However, there may be gaps ("padding") between variables.
- Padding may be controlled by a compiler parameter.
- Avoid writing or sending complete structs over networks: send them field-wise
- If you wish to compare structs, make sure to have initialized sizeof(mystruct) using memset, and only then you can compare them using memcmp.

C Floating point

- Existing standard: IEEE-695
- Most processors use 64 bits precision for temporary fp operations. Intel x86 and most of Motorola 68k use 80 bits.
- Optimization $\Rightarrow$ fused sequences of multiplication and addition in high precision. On heavy floating point programs this may cause a change in the flow of the program.

C Floating point- Solutions

- Most compilers have an option to disable the extended precision (for example '-ffloat-store' in gcc).
- Floating point programming should use epsilon values where required. For example, check for (fabs(x)>1.0e-15) rather than (x!=0), do not accumulate small numbers to a large accumulator.
- Check your software for FPEs (using compiler flags), but do not count on the compiler to always supply those mechanisms, requires by IEEE-695 standards (overflow, underflow, etc.)

Unix standards

Modern Unices comply with:

- POSIX.1 (1990 edition)
- POSIX.2 (1992 edition)
- SuS (The Single UNIX Specification, Version 3)

Functions missing in POSIX.1 may not be always available. (getuid, snprintf, mmap). Look for alternatives. Use autoconf. Check for existence using AC_CHECK_FUNCS in configure.in.

Time and Date

- The zero date changes (not always 0:00:00, January 1, 1970). Use ISO 8601 standard: YYYY-MM-DD.
- The type time_t, which time returns, changes.
- gettimeofday is not POSIX

Character sets

- Order of characters changes (ABC..abc, AaBbCc..., European vowels interlaced), even within the first 128 values.
- Not to mention existance of Hebrew charsets...
- Usages of ord(), chr() is not portable.
- isalpha from <ctype.h> will test for characters being of class alpha, given the locale.

making and configuring

- Avoid obsolete, not fully supported make syntax: for example, avoid double suffix targets like **.c.o:**. Use **%.o : %.c** instead.
- Avoid specific make extentions to syntax, for example gmake extentions.
- **Identify system functionality, instead of deducing from system names, thus supporting future systems.**
- Insert compiler parameters to configure.in, not to makefile.in .

Files

Not all operating systems have:

- Case sensitivity in file names.
- Directory separators (/ on Unix, : on Mac OS, \ on Windows, DOS).
- Spaces in file names (allowed (escaped)on Unix, but not customary– bugs are likely to appear when running on Windows).
- ":" in filenames is not allowed on Windows.
- Delimiter for lists of directories (Unix :, Windows ;) .
- Concept of a single root directory (/mnt/my_windows_machine vs. \\my_UNIX_machine\my_disk.

Files

Not all operating systems have:

- Support of renaming or unlinking open files $\Rightarrow$ Close files before doing suspicious things to them.
- Per program/process environment variables.
- Per program/process current directory.

Binary files vs. Text Files

or

The case of the infamous ^M

In Windows, text lines end with carriage return character followed by a line feed (\r\n).

- C programs which read a Windows text files will read two characters in binary mode (fopen with parameter 'b'), one in text mode.
- A C program which opens a text file cannot simply count characters and use the info for fseek.
- Ftp also needs to know the type of file.
- d2u, u2d (a.k.a. dos2unix, unix2dos) convert formats.

Shell Scripts

Shells themselves were ported to the machine already. Using the local shell might solve some porting issues. For example, using shell tools through embedding **system("my shell command");** within a C code. There are even shells for Windows.

- "#! /bin/sh " is the default shell on the system.
- The default shell usually resembles Bourne Shell.
- Usage of standard Bourne shell is usually implemented in all versions. (For example: '"**PATH= "/my/path": $PATH** " does not always work, do use **export**.)
- Sometimes there exists another shell with capabilities the default one lacks (for example, functions).

Command line tools

- Location of command line tools: specifying the location saves time and verifies the correct tool is used. On the other hand, when the location is not the same on relevant systems, not specifying the location may solve the problem.
- Verify and match : tool –version with man tool and info tool. locate tool to find other versions.

Perl Scripts

- Perl is a portable way for doing shell scripts.
- Perl is not always installed in the same place (/usr/bin, /usr/local/bin): use "#!**/usr/bin/env** perl" (env usually is located in the same place)

Readings

- The Autobook: http://sources.redhat.com/autobook/ by Gary V. Vaughan, Ben Elliston, Tom Tromey and Ian Lance Taylor
- Perlport: Writing Portable Perl
  http://www.perl.com/language/newdocs/pod/perlport.html
- Motivation for writing portable code
  http://www.byteswap.net/mikesnotes/2002/getting-started/
- Porting Guides
  http://www.unixporting.com/porting-guides.html

References

- PVM http://www.epm.ornl.gov/pvm/pvm_home.html
- tcl/tk http://www.scriptics.com/
- cygwin http://sourceware.cygnus.com/cygwin/
- pdf http://www.adobe.com/products/acrobat/adobepdf.html
- Gnu Tools for Windows: http://www.mingw.org/ http://www.delorie.com/djgpp/
- Using __attribute__ in gcc and other C compilers http://www.unixwiz.net/techtips/gnu-c-attributes.html
- http://www.unix-systems.org/version3/