Windows portability for GNOME software

Tor Lillqvist

tml@iki.fi, tml@novell.com

February 27, 2006

Novell®

Why port GNOME software to Windows?

- “Because it's there”. It's an interesting challenge
- Some people want it
- Might make Windows users interested in running such apps on the real thing instead

General

- Many applications written for GNOME can be built and run on Windows
- Some simple portability rules must be followed
- Some applications it just wouldn't make sense to port though, even if possible
- Ignore Win9x, please

Compiler and tools: MinGW

- ”Minimalist GNU for Windows”, but that's a bit misleading
- MinGW = gcc + binutils + reverse engineered headers for the Win32 API and Microsoft C library
- gdb port a bit buggy, but usable
- MSYS = POSIX shell and utilities like make, awk, sed, m4 and Perl needed to run auto* and configure scripts, and as interactive shell

Not MSVC?

- Can not use Microsoft's compiler because of deep technical issues in how ORBit2 and IDL-compiler -compiled code imports variables from DLLs.
- When linking to libORBit2, or IDL-compiler generated code, must the GNU linker with its `--enable-auto-import` and `--enable-runtime-pseudo-reloc` switches
- Issue with C runtimes: Only MSVC6 supports the bundled C runtime msvcrt.dll

C library

- C library: The bundled msvcrt.dll. C89 + a few POSIXish additions
- open() (but don't use, see *File name character set*), read(), write(), dup()
- File descriptors (the small numbers returned by open() and fileno()) are implemented in the C library. They are not known by the kernel
- <dirent.h> (opendir() etc) is a MinGW extension, but don't use, use GDir instead (see *File name character set*)

C library, continued

- wchar_t is 16 bits (one UTF-16 ”word”)
- all functions that take string arguments have wide character string counterparts: _wfopen(), _wstat() etc
- wide character string functions wcslen(), wcschr(), wcscpy() etc
- multi-byte character (system codepage) string functions _mbslen(), _mbschr(), _mbscpy() etc

C library, continued 2

- setlocale(LC_ALL, ””) does not look at any LC_* or LANG environment variables
- setlocale() uses ”English_United States.1252” -style locale names
- GTK+ and GLib do look at LC_ALL, LC_CTYPE and LANG
- To get the process's locale like on Unix, call g_win32_getlocale(). Returns a ”sv_FI” style string

C library, continued 3

- Text file normally have CRLF line endings, but just LF works, too
- Open files in binary mode in general.

```
g_fopen(filename, ”rb”)

#ifndef O_BINARY
#define O_BINARY 0
#endif
g_open(filename, O_RDONLY|O_BINARY, 0)
```

GLib

- Always use GLib functionality if available
- Prefer g_file_test() to stat() or access()
- g_mkdir_with_parents()
- g_get_file_contents()
- GmappedFile
- GDir
- UTF-8 collation functions
- Do add GLib feature requests to bugzilla

POSIX functionality

- No fork()
- No link(), lstat(), symlink(), realpath()
- No fsync()
- exec() exists, but just spawns a child and exits once the child has finished

Win32 API

- Documented online at msdn.microsoft.com
- Also documented in the freely downloadable Platform SDK, much quicker to read locally
- Huge number of functions
- Parallel APIs for system codepage (”ANSI”) and wide character strings: GetUserNameA() vs. GetUserNameW()
- Most of the wide character API not implemented on Win9x
- Usually very few, if any, Win32 API calls needed

Threading issues

- Don't call GTK functions from several threads
- Unlike the X11 protocol, the Win32 windowing and graphics API is thread-aware
- Windows knows what thread created a window, and messages (“events”) for that window are delivered to that thread's event queue, etc
- This all means horrible breakage if you create window or do windowing API calls randomly from different threads

pthread API

- A Free POSIX thread implementation available from SourceWare: pthreads-win32
- A lightweight and efficient wrapper around the native thread API. Works fine
- Standard POSIX thread API
- Portability: pthread_t is a struct! One cannot compare pthread_t values directly. No special ”NULL” pthread_t value. (Ditto on HP-UX.)
- Use pthread_equal() to compare pthread_t values
- Preferrably, use GThread instead of pthreads

File name character set

- File system uses Unicode (UTF-16)
- Each machine has a fixed ”system codepage”: a single- or variable-length (double-byte) character set
- Single-byte codepages: CP1252 etc. For European, Middle East languages, Thai, etc
- Double-byte codepages: In East Asia
- It's quite possible to have file names on a machine that can't be represented in the system codepage. Occurs in East Asia, and for Western Europeans who exchange documents with Greece, Russia, etc

File name character set, continued

- All file name APIs in the C library have two versions:
  - normal one (fopen) uses system codepage,
  - the wide character one (_wfopen) uses wchar_t
- But, forget all the above, just use UTF-8 and GLib
- GLib and GTK+ APIs use UTF-8
- gstdio wrappers for UTF-8 pathnames: g_open(), g_fopen(), g_dir_*(), g_stat() etc

File name character set, continued 2

- Illegal characters in file names: `< > | * ? :`
- Case insensitivity: Hard if you want 100% emulation of what the system would do:
  - Each NTFS volume has a case-mapping table that maps single wide characters to single upper case equivalents
  - Cases like ß ~ SS or precomposed ~ composing diacritic sequences not handled
  - Just don't bother

File name character set, continued 3

- Other libraries like libxml2 and gettext don't expect UTF-8 pathnames
- Need to pass them system codepage filenames
- g_win32_locale_filename_from_utf8() should work in most cases for existing files. It looks up the short (8.3) form of the name which always is in ASCII
- 8.3 name generation might be off on a volume
- g_locale_from_utf8() only if representable in system codepage

Pathname manipulation

- Always use Glib functionality:
  - g_path_get_basename(), g_path_get_dirname()
  - g_build_filename()
  - g_path_is_absolute(), g_path_skip_root()
  - G_IS_DIR_SEPARATOR()
- Search paths (PATH, BONOBO_ACTIVATION_PATH etc) use semicolon separator (G_SEARCHPATH_SEPARATOR)

file: URIs

- Don't confuse URIs and file pathnames
- `file:///X:/some/where/foo.bar`
- `file:////server/share/dir/sub/f.ext`
- Don't just prefix a filename with "`file://`"
- Don't just strip off a "`file://`" prefix
- Use g_filename_to_uri(), g_filename_from_uri()
- A relative pathname is not a URI. There is no such URI as `file:foo.bar` Just use the filename for relative links

Socket API

- `#include <winsock2.h>`
- For IPv6 and misc other additional stuff:

  `#include <ws2tcpip.h>`
- Sockets are not file descriptors. Sockets and fds even overlap! The same number can be both a socket and fd
- Cannot read(), write(), close() sockets. Those are C library functions. C library knows nothing about sockets
- Use recv(), send(), closesocket(), ioctlsocket()

Socket API, continued

- Can select() only on sockets
- Functions return SOCKET_ERROR on failure, but that is -1, so just checking for <0 like on Unix works
- error code after socket API calls not set in errno! errno is in the C library. Use WSAGetLastError() and WSAE* codes
- No UNIX domain sockets

Socket API, continued 2

- Best to use simple wrapper macros to hide the differences

```
#ifndef G_OS_WIN32
# define SOCKET_ERROR_CODE() errno
# define SOCKET_CLOSE(fd) close(fd)
# define SOCKET_ERROR_IS_EINPROGRESS() (errno==EINPROGRESS)
# define SOCKET_ERROR_IS_EINTR() (errno==EINTR)
#else
# define SOCKET_ERROR_CODE() WSAGetLastError()
# define SOCKET_CLOSE(fd) closesocket(fd)
# define SOCKET_ERROR_IS_EINPROGRESS() \
 (WSAGetLastError()==WSAEWOULDBLOCK)
# define SOCKET_ERROR_IS_EINTR() 0 /* No WSAEINTR errors */
#endif
```

Socket API, continued 3

- Would be best if the Unix/Winsock differences were wrapped by a library and its headers
- There are several more or less generic networking libraries, but unfortunately, none is ideal: GNet, libsoup, linc2 (in ORBit2), ...
- Use g_io_channel_win32_new_socket()
- Win32 implementation of watches on GIOChannels for sockets changed radically in 2.8
- g_io_add_watch()ed sockets automatically become non-blocking!

Spawning processes

- Use g_spawn_*() API instead of pipe()/fork()/dup()/exec() acrobatics
- Internally C library uses CreateProcess() which passes a command line, not an argument vector
- C library startup code reconstructs an argument vector from command line
- Quoting funkiness: g_spawn_* tries its best, but if possible avoid passing hairy arguments with spaces, backslashes etc

GUI and console apps

- An executable (EXE) is either GUI or console. This is just a flag in the header
- Console apps always run with a console window, either the one started from (“Command Prompt”), or open one automatically if started from Explorer or the Start Menu
- stdin/out/err normally attached to this console window unless redirected
- GUI apps normally have stdin/out/err pointing nowhere, and no way to print to the console window the were started from (if any). Redirect to a file or pipe to see printf output

DLLs and -no-undefined

- DLLs (and EXEs) can not have undefined symbols
- Always use -no-undefined when building shared libraries with libtool
- Do use DLLs whenever you use shared libraries on Linux
- Don't build static libraries unnecessarily. DLLs work fine and are very normal in Windows
- No separate LD_LIBRARY_PATH. PATH is used to search DLLs, too

DLLs and -no-undefined, continued

- Evolution has a complex mess of even circularily dependent shared libs
- The solution was to use separately built dummy “bootstrap” import libraries as stand-ins for import libraries for DLLs not yet built

Relocatability

- Windows software should be installable by end-user in any location
- One can't assume **anything** about pathnames on the end-user machine
- Software might be installed on a server in a UNC path that doesn't even have a drive letter
- Machine might not have a C: drive
- Pathnames might contain spaces or random Unicode characters

Relocatability, continued

- DLLs and EXEs can look up their location at run-time
- Lots of examples of this in GNOME libs, e-d-s and evo
- Macros like FOOBAR_GLADEDIR, FOOBAR_LOCALEDIR typically re-#defined in a header as function calls for Win32
- Paths to files needed at run-time then constructed at run-time

Relocatability, continued 2

- In a DLL: DllMain() is called when the DLL is attached to a process. Saves the DLL handle
- When constructing a pathname at run-time, the location of the DLL is looked up using its handle and the pathname is constructed
- Assume normal DLLs are in *prefix*`/bin` where *prefix* is the end-user installation prefix
- Strongly advice end-users never to copy DLLs around as an attempt to fix problems
- Never install anything in the system32 folder

GNOME platform and desktop libraries

- All those required by Evolution have been ported
- Seem to work OK to the extent required by Evo
- GnomeVfs: just basic functionality
- ORBit2: no Unix domain sockets

Case: Evolution

- Port took 7—9 months
- Half of the effort spent on porting the dependencies
- Available from ftp.gnome.org
- No installer generally available yet
- All Win32 changes in CVS and GNOME 2.13 etc tarballs

Novell®