WINE- The Windows Emulator

Marcus Meißner <meissner@suse.de>

History

The WINE project was founded by Eric Youngdale and Bob Amstadt in 1993 with the idea of creating a binary emulation for Windows 16 bit programs in Linux. Soon it was able to run the first windows programs, including solitaire, winmine and notepad. The last 'usual' numbered release was 0.8 in January 1994, after that WINE releases were labeled after the date.

The newsgroup comp.emulators.ms-windows.wine was created as a forum in mid 1994. In 1995 the implementation of the Win32 API was started, making it possible to run the first programs of Windows NT 3.51 and Windows 95 in 1996. 1996 also saw Excel and Word (both the 16 bit versions) reportedly running.

While development was continuos, it was at times slow. Nevertheless the project saw the first implementation of DirectX in 1997 and a lot of Win32 programs already starting up in 1998. 1997 also saw the creation of the WINE website http://www.winehq.com/ and the associated developer mailinglists.

A major push to the speed of WINE development came in the end of 1998 as Corel decided to port its Office suite to Linux using WINE. During the development lots of resources were pushed into the WINE project to enhance areas like printing and common controls which were lacking for Corels purposes. While originally it was meant to be a real sourcecode port of the Office Suite, however this turned out to be too difficult, so a special compiled binary was run in normal emulation mode. Corel released WordPerfect Office 2000, Corel Draw 9 and Corel Photopaint 9 based on WINE in 1999, but discontinued this line of work due to monetary problems.

Nevertheless companies grew out of this period, namely the Canadian company Macadamian (http://www.macadamian.com/) and the Minnesota based CodeWeavers (http://www.codeweavers.com/).

In 2000, Borland used WINE to port their Delphi Rapid Application Development suite to Linux, under the name of Kylix.

CodeWeavers today is overseeing the project and sponsoring its infrastructure, no to mention employing several key WINE developers, including the WINE chief Alexandre Julliard himself.

While CodeWeavers is concentrating on Office products, the Canada based company TransGaming (http://www.transgaming.com/) started focusing on the games support.

Last notable event was the first ever WINE Developer Conference that was held by Lindows (http://www.lindows.com/) in March 2002 in San Diego, California. A lot of WINE developers first met each other in person at this conference.

The project itself is lead by Alexandre Julliard since 1994, who also is the only one that has write access to the CVS repository of the WineHQ tree. The developers are spread all over the globe and only communicate by E-Mail.

Licenses

An interesting, however a bit not so funny, point in WINE development are the various license changes and their reasons. Wine has gone over a lot of licenses during its time, if not to say nearly all of the better known.

It started as a GPL licensed project, with the decision coming from the Linux kernel background.

In 1994 the license was changed to BSD to allow companies to take the code and use it for their own porting purposes to migrate their users to Linux. That they might not return their code was accepted by the developers at this time, since it would not stop the developers from doing the work themselves. In 2000 the license was changed to X11 to avoid using the BSD license disclaimer clause.

2000 also saw the founding of TransGaming (http://www.transgaming.com/), a company whose goal is to bring Windows games to Linux, by using WINE and enhancing DirectX and other game related parts of it. Their business model is subscription based, and their source tree is open for viewing, although based under the Aladdin Free Public License (AFPL). This license allows private use of the code, but restricts redistribution and reuse severely. With DirectX enhancements available from this tree for 'free', the development of DirectX in the OpenSource tree effectively stopped. Also as TransGaming begun to work on non-game related areas (namely the support of the InstallShield v6 installer), the unrest in the rest of the WINE community grew in regards of TransGaming taking things out of the free tree and only contributing 'non essential parts' back, effectively leeching of the project.

Long discussions and flamewars later the license of the free tree was changed to LGPL, which reflects the intent of most developers. TransGaming in turn created a new X11 licensed fork, called ReWind (http://rewind.sourceforge.net/). This tree is mostly inactive at this time, also showing the preference of most developers to go with the LGPL license.

Technical Overview

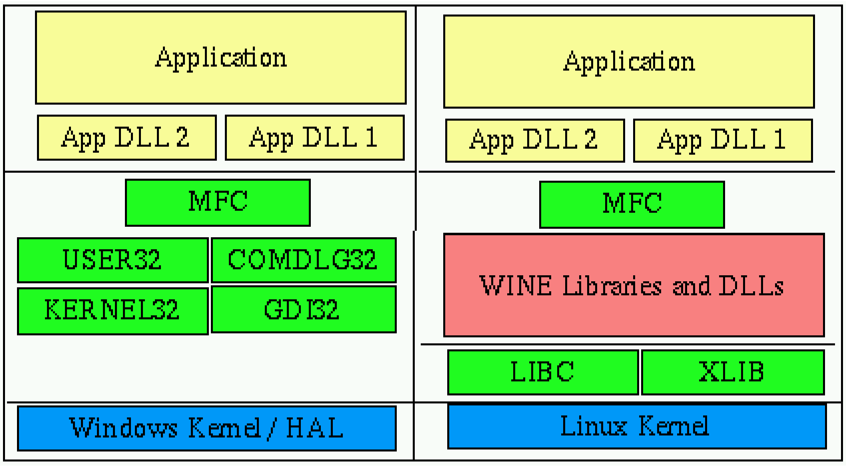
WINE in the traditional sense of recursive acronyms stands for 'WINE is not (just) an Emulator', or also 'Windows Emulator'.

The reason for this is that WINE does not emulate everything down to the processor, it just replaces the lowlevel APIs of Windows.

Following figure shows how WINE fits into the executables and library structure on both Windows (left side) and Linux (right side):

WINE itself does not replace the kernel layer of Windows, it just replaces the core libraries, like KERNEL, USER and GDI. DLLs on top of those can either come out of WINE or get reused from Windows. The application however gets installed as is, including its DLLs. Since WINE does not reimplement the kernel layer of Windows, it is not possible to use WINE to support Windows kernel drivers (like WinModems, printers, USB devices, or similar) under Linux.

WINE does not start the full Windows system, it just starts applications and just provides a virtual Windows environment to them to integrate fully into the normal Linux system. This is the difference between virtual PC emulators, like VMWare or BOCHS and WINE, while the former emulate the whole PC including the BIOS and Windows startup code and all hardware, WINE does not.



Loading Windows Executables

WINE uses a small loader part that call the WINE implementations of Windows program start functions on its argument. These functions use mmap(2) to map the Windows .exe and .dll files into the address space of the UNIX process, resolve pending relocations and imports and load referenced WINE .dlls. The process loader then sets up the rest of the address space, the stack, environment variables, process and threading data structures and the main thread of the just loaded process gives control to the startup function.

On exit the respective structures are cleaned up and deallocated.

KERNEL Functionality

The following Win32 API components are mapped to their UNIX equivalents.

- Win32 Processes: These are just mapped 1:1 to UNIX processes.
- Win32 Threads: These are mapped 1:1 to the kernel threads on the respective platform. Under Linux WINE uses clone(2) threads. Pthreads have been considered, but these are too high-level for the purposes WINE needs.
- Synchronization primitives: The Win32 API features a lot of primitives for synchronizing threads and processes, including Critical Sections, Mutexes, Semaphores, Events, Named Pipes, Mailslots etc. All these can be synchronized together (like wait for an event to happen on any of those types).

  This is rather hard to implement using standard UNIX system calls or library functions so WINE uses a standalone process (the 'wineserver'), which handles waiting on those objects.
- File System Access: The filenames exposed to the Win32 API are of course the standard DOS filenames with drives and backslashes in them. Those are remapped to the UNIX filenames using drives specified in the configuration file, by using a mapping mechanism, which also honors the fact that the usual Windows filesystem preserves the case of the filename, but allows case insensitive access to it. Also required is mapping between short and long filenames, for the Win16 API only supported short filenames. The mapping functions are some of the most difficult pieces of code in the WINE tree.

- Memory Management:

  Memory management is implemented exactly as in Windows on top of mmap(), malloc() or sbrk() style memory allocation is not used. Reason for this is that some programs try to access the low-level memory management and will get confused by the normal UNIX malloc() process. Also the locking primitives the latter uses are not compatible with the WINE locking primitives which might lead to memory corruption.

- Win32 Console: The Win32 text console is implemented using a standalone terminal/console program, which communicates with the calling program and the wineserver process.

- Thread Local Storage: One of the more difficult points in the Win32 thread API is the local storage, which is referenced by the 'FS' segment selector register. (Yes, even in 32bit mode, selectors are used.) This is used to easily access thread information, and to implement a lightweight exception handling framework called structured exception handling.

- Registry: WINE implements the full Windows registry API, including the ability to load the native registry format of Windows 3.1, 95, 98 and 2000. The registry is saved in text format, so it can be handled the UNIX way. A registry editor is included as a sample program.

Graphical Functionality

Most of the Windows graphical components are mapped to the X11 equivalents.

- Graphical primitives: These are, if possible, implemented using the Xlib functions. Some functions that take a lot of effort, like blting bitmaps is optimized using the X shared memory extensions and similar.

- Text Output: The text output engine of WINE uses either the built-in X fonts, or if FreeType and the X Rendering Extensions are available, the latter to output text.

- Printing: WINE implements only 1 printer driver, which outputs PostScript. It is configured using the PPD (PostScript printer description) file, which is either specified in the configuration file, or retrieved using the CUPS printing system (if available).

  For printing WINE is now capable of printing with TrueType fonts, it embeds them into the generated PostScript file.

  The printing itself is done using by calling the external "lpr" program, with the list of available printers supplied by either CUPS or read from /etc/printcap.

- Window handling: The windows displayed are normal X11 windows, with accompanying hints for the window manager. Windows messages can be sent to windows of different processes by again facilitating the wineserver process.

  The windows themselves can be either managed (by the window manager itself) or unmanaged (by WINE, where WINE draws the borders and decorations by itself). While the former ensures a better integration, the latter is more compatible to when the windows program might draw to the window decoration itself.

- DirectDraw: The 2d component of DirectX, DirectDraw, is implemented again on top of the Xlib toolset, but uses as much speed up features as possible, including shared memory pixmaps or even DGA mode if possible.

- OpenGL support: Since the ABI of OpenGL is the mostly the same under Windows and under UNIX only some remapping needed to be done to support it to its fullness. It is possible to just play any OpenGL windows game with full hardware support.
- Direct3D: The 3d component of DirectX is only implemented to a useable degree in the TransGaming version, WineX. It is implemented on top of OpenGL and so also facilitates the full hardware acceleration possibilities.

Hardware Access

- Networking Support: The networking API, that implement the standard TCP/IP and UDP/IP communication are implemented on top of the UNIX networking APIs. So there is no need to configure a dial-in network in WINE, it just reuses the one from the running System. There is also some IPX support for older network games.
- Sound: The various Sound APIs are implemented on top of the Open Sound System and several others. For DirectSound, the sound component of DirectX, full speed support using mmap'ing of sound devices and similar were implemented.
- Input: Keyboard, Mouse and Joystick input is supported using X11 methods, or using the Linux joystick device respectively.

Uses of WINE

Wine can be used to just run Windows programs. However, other methods of use are also thinkable:

- Source porting: WINE supplies header and development tools to compile Windows applications natively under Linux making migration of source code easier.
- Using Windows DLLs: Some times there are Windows video or audio compression codecs which are not available under Linux. These can be reused in Linux by using WINE to load and run them.

  As some of the well known examples, avifile and MPlayer both use a reduced set of WINE code to load Windows MCI codecs to decode and play AVI files, including DivX movies. A different example uses WINE to load winamp plugins for XMMS.
- As browser support: A KDE project called reaktivate was started but not finished, which was used to implement ActiveX support in the KDE web browser konqueror. It was working rather well.

Current Status

WINE somehow has the impression of an ever lasting Alpha project. Actually 3 years ago as I held the same lecture I hinted that a final release might only be half a year if not one year away. 3 Years later I can say that in probably half a year we might start beta snapshots. The WINE conference in San Diego again listed the goals required for WINE to go in Beta Status and several of them are in progress if not already finished. So the Beta status is not that far away.

Speed of development has been varying over the years, but with the involvement of commercial companies it has been steady if not increasing over the last years.

As for the status of implemented libraries, most of them are feature complete.

When WINE 1.0 is reached, a lot of things still need to be done, but the core will be finished and useable.

As for applications, most applications start and run to some degree. Even large applications like Lotus Notes only need some small adjustments

A list of applications, maintained by the WINE users, can be found on http://appdb.codeweavers.com/.