Wine Is Not an Emulator

Shachar Shemesh
Open Source Consultant
http://shemesh.biz

Thanks to Netta, Jess and Amos for not letting me concentrate on these slides

(Netta did let me)

History

```
Newsgroups: comp.os.linux
(From: bob@amscons.com (Bob Amstadt
Subject: Re: Impressions of WABI/Univel vs
Linux
<References: <C9JoGx.IMM@news2.cis.umn.edu
Organization: Amstadt Consulting Group
Date: Fri, 2 Jul 1993 17:36:48 GMT
X-Newsreader: Tin 1.1 PL4
Message-ID:
<<1993Jul2.173648.3202@amscons.com
Lines: 7
Of course, for those of you who are
interested in running Windows programs
there is an effort in progress to create
.something similar to Sun's WABI
There is an activists channel, "WABI",  for
  .discussion of this project
There is need for many volunteers to complete
.this project
 --
Bob Amstadt
bob@amscons.com
```

Historical Details

- Founded in 1993, by Bob Amstadt
  - Long before Windows 95 made Win32 the de-facto standard.
  - Aimed at supporting the Windows 3.1 API
  - Had lots and lots and lots of work ahead

Where Have We Progressed To?

```
To: wine-announce@winehq.com
Subject: Wine release 20030618
From: Alexandre Julliard <julliard@winehq.org>
Date: Wed, 18 Jun 2003 14:35:27 -0700

This is release 20030618 of Wine, a free
implementation of Windows on Unix.  This is
still a developers only release.  There are many
bugs and unimplemented features.  Most
applications still do not work correctly.
```

The Theory of General Relativity

No matter how much you work, there is still much more work ahead of you.

The Rocky Road

- Wine started of as a X11 license project.
- Three major companies were involved in Wine development.
  - "CodeWeavers" – distributing a proprietary product called "Cross Over Office" – aimed at making Microsoft Office installs an easy thing.
  - "Transgaming" – distributing a proprietary product called "WineX" – aimed at making games run simply and easily.
  - Originally "Lindows" – distributing the "run your Windows app on our distro" product by the same name.

The Rocky Road (cont.)

- Late 2001 a group of Wine developers, as well as CodeWeavers, felt that Transgaming was breaking the social contract.
- Copyright holders were located, and the license was changed to LGPL.
  - A mailing list called "wine-~~flame~~license" was formed
- Somewhere along the road, Lindows abandoned all Wine related hopes.

Current Wine Deriviatives

- Winehq (http://winehq.org) – The main (LGPL) wine branch.
- ReWind (http://rewind.sf.net) – A fork branched from the last X11 licensed version. Mainly there for TransGaming to use.
- CodeWeavers (http://codeweavers.com) – Sell XOO, as well as other goodies (cross over plugin) – based on the LGPL version.
- Transgaming – Distributing WineX – based on ReWind.
- ReactOS (http://www.reactos.com) – a Windows NT compatible OSS project.

So What IS Wine?

- It's a free software reimplementation of the Windows API.
- Wine is NOT an emulator.
  - Honestly.
- Supported platforms:
  - Linux Intel
  - Solaris x86
  - FreeBSD
- Early stage development platforms
  - Windows
  - Mac OS/X – integrated with an emulator there.

The m-w dictionaty: Emulator

Main Entry: em·u·la·tor

- Pronunciation: 'em-y&-"lA-t&r
- Function: noun
- Date: 1589

  1) one that emulates
  2) hardware or software that permits programs written for one computer to be run on another usually newer computer

Case in point – an Emulator simulates a *hardware* platform.

Proof The Wine is Not an Emulator

```
/***********************************************************************
 *      CreateWindowExW (USER32.@)
 */
HWND WINAPI CreateWindowExW( DWORD exStyle, LPCWSTR className,
                                 LPCWSTR windowName, DWORD style, INT x,
                                 INT y, INT width, INT height,
                                 HWND parent, HMENU menu,
                                 HINSTANCE instance, LPVOID data )
{
    ATOM classAtom;
    CREATESTRUCTW cs;
    WCHAR buffer[256];

    if(!instance)
        instance=GetModuleHandleW(NULL);

    if(exStyle & WS_EX_MDICHILD)
        return CreateMDIWindowW(className, windowName, style, x, y, width,
height, parent, instance, (LPARAM)data);

    /* Find the class atom */

    if (HIWORD(className))
    {
        if (!(classAtom = GlobalFindAtomW( className )))
        {
            ERR( "bad class name %s\n", debugstr_w(className) );
```

The Challange

- OS supplied DLLs
- Fonts
- "Forest" filesystem approach
- Registry
- Services
- DirectX
- COM/DCOM
- MFC
- Device drivers
- Boot sequence
- 16/32 bit interfaces
- "Native NT" interfaces
- Windows messages

And How is Wine Doing?

Pretty good, actually!

http://winehq.org/?page=status – *The "Wine Status" Page*

- Tries to list what has been done.
  - Sometimes falls out of date
- In general – Many many many aspects of Windows have been touched to some degree.
  - Most applications can be made to work, to some degree.
  - My personal recomendation:
    "Wine is already good enough to replace specific applications, if those are all that blocks a move from Windows to Linux".

So Why the Alpha Status?

- http://www.winehq.org/?page=todo_lists lists the tasks required before a "beta" status can be announced
  - Setting up wine is still a matter of packaging.
  - When Wine is not yet installed, things are not functional enough to install it.
  - Too many applications require "magic" to get them to work.
- Hoping to move to "beta" status, but no clear date yet.

Wine Internals

What's inside the box

Under The Hood

- Wine has the following major components:
  - Win32/16 API implementation
    - By far the greatest part of the project. Reimplementing the various API calls that make up the Windows environment.
  - Backend
    - Wine is not an OS. It uses various backends in order to actually perform the tasks.
    - Currently supported: X11 and tty.
    - In addition there are tons of "soft" and "hard" dependndancies.
  - Loader
    - This is simply a runtime linker that understands the PE, MZ and that third file type used by Windows.

PE Loader

- For the most part – just relocate a PE executable into memory, load dependant DLLs, and link the relevant entry points from one to the other.
  - PE DLLs – Native Windows DLLs. These cannot link to Unix shared objects.
  - ELF executables – cannot link to windows DLLs.
  - The solution – winelib.
    - An ELF creature that can also use the Wine PE loader to link with Windows DLLs.

Backend

- Provides the required functionality to the various implementations.
- Hard and Soft dependancies
  - Hard dependancies – libraries that, if present during compilation, must also be present during runtime.
  - Soft dependancies – libraries that, if not present during runtime, will reduce Wine's functionality, but not prevent it from running.
    - Most noteable example – presence of FreeType will allow use of local TrueType fonts.

The WIN32 Implementation

- By far, the most extensive part of the project.
- Recent project – DLL seperation
  - Make sure each Win32 DLL only depend on those DLLs it depends on in Windows.
  - Seperate the Win32 part and the actual implementation.
- All Wine DLLs are implemented as winelib apps.
  - Can be called by the PE DLLs, and yet call native Linux/Unix shared objects.

Wine Server

- There are many minor semantic differences between Win32, and the corresponding POSIX functionality. Especially in multi-process contextes.
  - Windows locks a file once it's open for reading or writing. Unix doesn't.
  - Windows has a registry, that is simultaniously accessable to all processes.
- To give Wine the Windows semantics, the wineserver coordinates the actions by all processes.

End User Experience

Installing Wine

Installing From Source

- Packaged install will not be covered here.
- Extracting sources the usual way.
- Either:
  - ./configure && make depend && make && make install
    - Only really an option for upgrades.
  - tools/wineinstall
    - Be prepared to grab a lunch (or two).
- On RedHat9 – make sure to include "--with-nptl", or wineserver won't start.

The Config File

- The "~/.wine/config" file controls how Wine behaves.
- When things don't work – that's usually the place to look for an answer.
- Almost all unix→windows conversions are done through this config.

Common Pitfals – Drive Mappings

- Windows uses a “forest” model.
- In order for Windows programs to know which drive they are on, the wine config holds a mapping of native directories to Windows drives.
- Tell sign:
  “Warning: L”filename” not accessable from a configured DOS drive”
  - Means that there is no Windows name to the unix name you tried to run.

Windows Management

- Managed mode – the X11 window manager manages the Wine windows.
- Unmanaged mode – Wine tries to manage it's own windows. The window decorations are the same as in Windows.
  - Sometimes this mode causes focus and Z order problems.
  - For some reason, it is the default under some RPMs
- Desktop – Wine wraps the entire window set with a “desktop” window.

DLL Overrides

- Why?
  - Some applications install their own versions of the DLLs. These sometimes don't work with Wine.
    - DirectX
  - Other need functions not yet implemented in Wine.
    - MFC
- DLL Overrides provide a way for the user to choose which version is used.

DLL Overrides – How

- In the config file:
  - Section called “[DllOverrides]”
  - builtin – the winelib version that came with Wine.
  - Native – the PE native version.
- Specify order
  - Item is the DLL name
  - Value is of the form “b,n”.
- Can also define a per-applicatoin override
  - Section [Appdefaults\\appname\\DllOverrides]

Install Types

- Fake windows
  - There is no real windows to rely on.
  - At most, you copy some DLLs from a real windows, and use DLL overrides.
- Windows based
  - The “C” drive is pointing to the Windows mount point.
  - The original DLLs and registry are used.
    - Using a read/write share has been known to corrupt the windows drive
    - Was the recommended method of install in the past, but is no longer seriously tested.

Fonts

- Wine will use the X11 fonts only if it really really really has to.
  - Either no FreeType is available (compile or run time).
    or
  - There are NO directly available fonts.
- *One* directly available font means that the X11 fonts will no longer be available.
  - This can happen rather stealthily.
  - Hebrew does not work well in X11 mode.

The Israeli Angle

- Hebrew is only very partially implemented.
- Right now, TextOut is the is only part really implemented.
- Edit control, Menu Items, Dialogs, and many many many other areas are not yet done.
- About 5% of the work is done
  - Covers about 80% of the functionality needed.
- Currently uses ICU to do the reordering.
  - FriBiDi was considered, but uses the wrong encoding.
  - ICU was almost thrown out for it's interface.