„Kompatibilität“ - Windows-Programme unter Linux laufen lassen und umgekehrt?

1 Allgemeines

„Linux ist nicht Windows“ (generelle Informationen und Einführung in die Thematik verschiedener Betriebssysteme, Technik und etwas Philosophie:

http://www.felix-schwarz.name/files/opensource/articles/Linux_ist_nicht_Windows/

Verschiedene Betriebssysteme funktionieren unterschiedlich und verwenden unterschiedliche Technologien, daher laufen grundsätzlich Binärprogramme (Maschinencode) nur auf der Hard- und Software, für die sie ursprünglich geschrieben worden sind.

2 Emulationen / APIs

„Emulatoren“ stellen einer Software ihre „gewohnte Umgebung“ (API) zur Verfügung, so dass sie auf einem anderen Betriebssystem als dem, für das sie erstellt wurden, lauffähig wird.

Wine

(„Windows-Emulator“) stellt eine Microsoft Windows-kompatible Umgebung her, so dass Windows-Programme lauffähig werden, vorausgesetzt, es sind alle notwendigen „Windows-Komponenten“ installiert. Viele Programme erwarten ihre Laufzeitumgebung in Form von sogenannten „DLL“-Dateien (entspricht Bibliotheken unter Unix).

Es gibt speziell für den Wine-Emulator Kompatibilitätslisten, die angeben, welche Windows-Software damit läuft → http://appdb.winehq.org/.

(X)dos(emu)

Erlaubt das Starten von „DOS“-Programmen unter Linux (auch graphische DOS-Programme/Spiele), wobei im Hintergrund ein „echtes“ DOS als Open Source eingesetzt wird. → http://de.wikipedia.org/wiki/DOSEMU

UAE

„Useless Amiga Emulator“, simuliert einen Commodore Amiga, und lässt die Amiga-Programme darauf laufen. Dies ist schon fast eine „Virtualisierung“, da der Amiga eine andere CPU und Architektur besitzt. → http://de.wikipedia.org/wiki/Amiga-Emulator#UAE

Spielkonsolen-Emulatoren

scummvm – Adventures

Für fast alle Spielekonsolen gibt es Interpretersprachen, die es erlauben, die Spiele auch ohne entsprechende Spielekonsole unter anderen Betriebssystemen laufen zu lassen, allerdings ist dies nicht in jedem Fall legal (Stichwort „Kopierschutz“). → http://de.wikipedia.org/wiki/Scummvm

3 Virtuelle Maschinen

Virtuelle Maschinen, oder „Voll-Virtualisierung“, simulieren eine komplett andere Hardware, als im Rechner tatsächlich vorhanden ist, wobei die Maschinen-Anweisungen der Programme allerdings teilweise auch „nativ“ auf der echten CPU und Peripherie ausgeführt werden können („Paravirtualisierug“).

Hiermit werden NICHT einzelne Programme eines anderen Betriebssystems ausgeführt, sondern das „Gast-Betriebssystem“ tatsächlich vollständig gestartet. „Rechner im Rechner“.

Vmware (proprietär), mit graphischer Administrationsoberfläche,

Virtualbox (Open Source) , mit graphischer Administrationsoberfläche,

qemu/kvm (Open Open Source), Kommandozeilen-basiert, Oberfläche des Gast-Betriebssystems ist natürlich graphisch,

kvm ist fast identisch mit qemu, führt aber viele Befehle direkt auf der Hardware aus, und ist deswegen performanter.

xen (Open Source), eine Virtualisierung, die im Linux-Kernel „eingebaut“ werden kann, und die fremde Betriebssysteme quasi als „Programm“ startet.

Vmware und qemu gibt es auch für Windows. Hiermit ist es möglich, Linux und Linux-Anwenungen in einem „virtuellen PC“-Fenster unter Windows laufen zu lassen.

Der „simulierte“ Rechner ist, abhängig davon, ob CPU und Virtualisierugssoftware „Paravirtualisieurung“ beherrschen, zwischen 40% bis fast 100% der Originalgeschwindigkeit des „echten“ Rechners kann erreicht werden.

4 Remote Services

Windows-Server / rdesktop

„rdesktop“ ist ein Client für Windows-Server, er benutzt das „RDP“-Protokoll und erlaubt es, sich von Linux aus graphisch auf einem Windows-Server einzuloggen, und die dortige Windows-Oberfläche zu nutzen.

VNC Server und Client

„Virtual Network Computing“ (oder besser „Visual ...“) ist ein System, mit dem der eigene Desktop auf andere Rechner „Projeziert“ werden kann. Im „Viewonly“-Modus kann der entfernte Desktop nur „beobachtet“ werden, ansonsten darf der Desktop auch über die Eingabegeräte des entfernten Rechners „bedient“ werden.

Beispiel:

Lehrer-Rechner (Server) mit IP 10.0.20.15: **x11vnc -shared -viewonly**

Studenten-Rechner (Clients): **xvncviewer 10.0.20.15**

Die Clients können jetzt den Lehrer-Rechner „beobachten“, und sehen, was dort passiert, aber wegen der „viewonly“ können sie in diesem Szenario keine Aktionen auf dem Lehrer-Rechner durchführen.

http://de.wikipedia.org/wiki/Virtual_Network_Computing

5 Kompatibilität vs. Plattformunabhängigkeit

Möglichkeit A: Die gleiche Software für verschiedene Betriebssysteme (und -versionen) anbieten.

Beispiele (Open Source): OpenOffice/LibreOffice, Firefox, Chromium-Browser, Java (Runtime) von Sun/Oracle.

Beispiele (proprietäre Software): flash-Plugin, Opera-Browser, vmware-Virtualisierung.

Möglichkeit B: Die Software mit einer betriebssystemunabhängigen Plattform herstellen (z.B. Java-Programme, Flash-Programe (proprietär), Browser-basierte Programme mit Javascript/Ajax).

Voraussetzung dafür, dass dies funktioniert, ist natürlich die jeweilige Laufzeit-Umgebung (Java-VM für Java-Programme, Flash für Flash-Programme usw).

6 Geschäftsmodelle proprietär vs. OSS

Priorität kommerzieller proprietärer Software-Verkäufer:

Viele Kunden, die Nutzungs-Lizenzen kaufen.

Beispiele: Spiele, Anwendersoftware mit herstellerabhängigen Formaten.

Priorität kommerzieller Open-Source Softwareentwickler und -supporter:

Viele (oder wenige, aber dafür besser) zahlende Kunden, die Anpassungen und Support oder Dienstleistungen (Zugang) kaufen.

Beispiele: Online-Rollenspiele, bei denen weniger das Programm selbst, sondern mehr die Community-Erlebnis im Vordergrund steht, Anwendersoftware mit offenen Standards.

In beiden Fällen ist gewünscht, dass möglichst viele die Software nutzen oder nutzen können, was die Motivation zur Schaffung von Interoperabilität erhöht.