QEMU – Virtuelle Computer für viele Betriebssysteme

Robert Warnke
Thomas Ritzau

QEMU

Virtuelle Computer für viele Betriebssysteme

QEMU Version 0.9.1

Deutsche Originalausgabe

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

Website: *http://qemu-buch.de*

Satz: Robert Warnke
Herstellung: Books on Demand GmbH, Norderstedt

ISBN-13: 9783837008760

Inhaltsverzeichnis

Einleitung

Manchmal benötigt man ein Programm, das eigentlich auf einem anderen Betriebssystem läuft. Oder auf dem Arbeitsrechner ist ein System installiert, das leicht angreifbar ist, man will aber trotzdem sicher im Internet surfen. Web-Designer müssen Websites auf unterschiedlichen Plattformen mit unterschiedlichen Web-Browsern testen, auch Entwickler benötigen unterschiedliche Testumgebungen. Oft hält man eine Präsentation beim Kunden vor Ort und möchte ihm eine Programm-Demo zeigen und muss dabei aber unabhängig von dessen Betriebssystem agieren können. Vielleicht will man auch nur ein anderes Betriebssystem testen.

Die Lösungen für diese Aufgaben heißen Virtualisierung und Emulation. Diese Techniken gehören nicht nur zu den Topthemen der IT-Insider, auch für Privat-Anwender werden sie immer interessanter, weil ein Betriebssystem zum einen nicht immer alle Bedürfnisse erfüllt und zum anderen moderne Hardware so leistungsfähig ist, dass auf dieser ein gleichzeitiger Einsatz mehrerer Betriebssysteme möglich ist. Diese Kombination bewirkte die rasante Entwicklung vielfältiger Lösungen im Server- und Desktop-Bereich, die das gleichzeitige Ausführen mehrerer Betriebssysteme unterstützen.

Eine dieser Lösungen ist QEMU. Das Programm ist im Internet zu beziehen unter *http://www.qemu.org* beziehungsweise *http://fabrice.bellard.free.fr/qemu.* Das kostenlose, quelloffene QEMU emuliert die komplette Hardware eines Computers mit CPU. Damit ist es möglich, auch Software, die für andere Prozessorarchitekturen vorliegt, auf dem Host auszuführen. QEMU ist nicht, wie zum Beispiel VMware, auf die x86-Architektur beschränkt. Darüber hinaus wird mit dem optionalen Beschleuniger KQEMU die Ausführungsgeschwindigkeit auf der x86-Prozessorarchitektur stark erhöht.

QEMU läuft auf vielen Betriebssystemen und Prozessor-Architekturen und lässt sich einfach installieren. Für die Installation von QEMU ohne KQEMU sind nicht unbedingt Administratorrechte erforderlich. So kann zum Beispiel eine virtuelle Maschine inklusive QEMU auf eine DVD gebrannt und auf einem anderen Rechner gestartet werden, ohne dass dort QEMU installiert werden muss. QEMU emuliert x86-, x86-64-, AMD64-, Power PC- und SPARC32/64-Architekturen. Weiterhin wird an der Unterstützung für Alpha, ARM, S390, IA-64, m68k und MIPS entwickelt.

Zum QEMU-Paket gehört auch das leistungsfähige Tool *qemu-img* zum Anlegen, Konvertieren und Verschlüsseln von Image-Dateien (virtuellen Festplatten) in unterschiedlichen Formaten, auch solcher anderer Virtualisierungssoftware. So ist es mit QEMU beispielsweise möglich, eine virtuelle Maschine für den kostenlosen VMware-Player anzulegen. QEMU selbst kann virtuelle Maschinen von Image-Dateien anderer Virtualisierungssoftware direkt starten. Das gleiche gilt für das Starten von Live-CD/DVD- und Boot-Disketten-Images. Mit QEMU muss dafür der Computer nicht neu gestartet werden.

Unter Linux und Mac OS X unterstützt QEMU auch die so genannte Userspace-Emulation. Diese ermöglicht es, dass ausführbare Programme, die für andere dynamische Bibliotheken kompiliert wurden, im Userspace betrieben werden können.

QEMU besitzt Funktionen, die andere Virtualisierungssoftware nicht bieten, wie zum Beispiel die Unterstützung beim Debuggen des Systems und das Booten von älteren Disk-Formaten. QEMU erlaubt es, Zustände der gesamten virtuellen Maschine mehrmalig zu speichern. Mit diesen so genannten VM-Snapshots lassen sich Änderungen, wie zum Beispiel das Einspielen von Software, rückgängig machen.

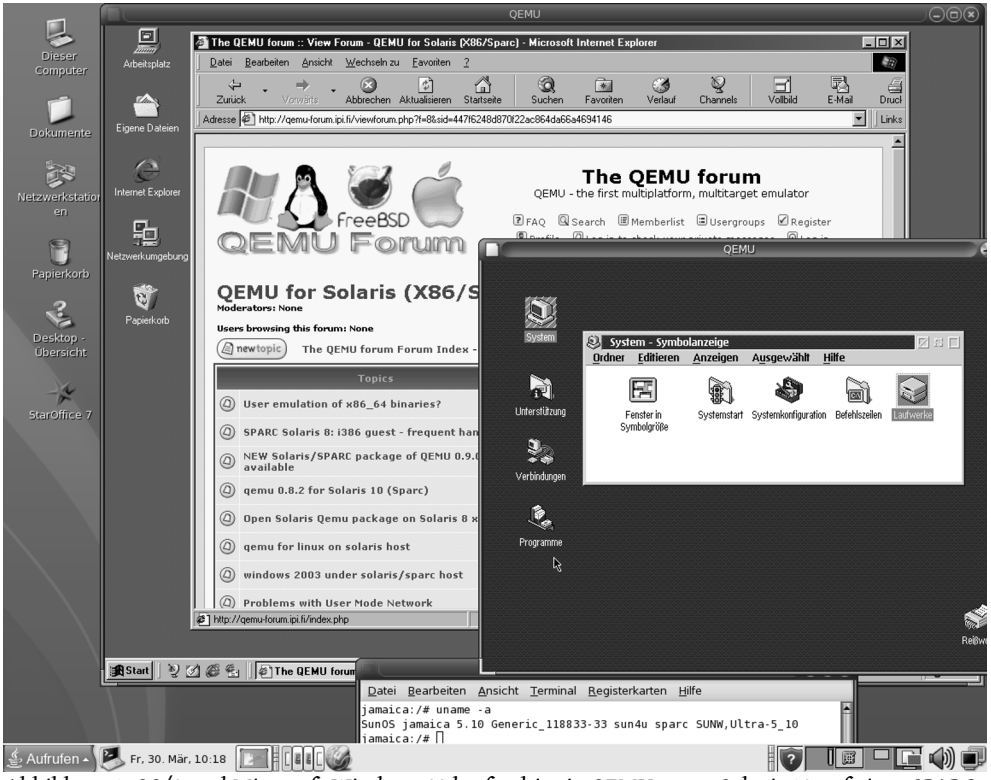

Abbildung 1: OS/2 und Microsoft Windows 98 laufen hier in QEMU unter Solaris 10 auf einer SPARC-Prozessor-Architektur.

QEMU wird von dem Franzosen Fabrice Bellard als Open-Source-Software entwickelt, die Quelltexte sind also frei verfügbar und können unter Wahrung des Urheberrechts und der jeweiligen Lizenzbestimmungen von jedermann angepasst und geändert werden. Die QEMU Virtual CPU Core Library (*libqemu.a*) und der QEMU PC System Emulator stehen unter der GNU Lesser General Public License (LPGL), während der Linux Userspace Emulator unter der GNU General Public License (GPL) veröffentlicht wird. Der optionale QEMU-Beschleuniger (KQEMU) steht unter der GPL, wobei aber die Headerdatei *kqemu.h* den Bestimmungen der BSD-Lizenz unterliegt. Trotz der komplizierten Lizenzbedingungen ist QEMU kostenlos!

QEMU nutzt Teile des Sourcecodes vom Bochs-Projekt, zum Beispiel das PC-BIOS. Viele andere Virtualisierungslösungen nutzen im Gegenzug Teile des Sourcecodes von QEMU, wie dies KVM und Xen beispielsweise tun. Dank der Offenheit des Sourcecodes und der Konfiguration mit Startoptionen ist die Entwicklung von Zusatztools unproblematisch. So entstanden zum Beispiel gute grafische Benutzeroberflächen zur einfachen Bedienung von QEMU. Auch kann QEMU leicht anhand von Scripten verwaltet werden.

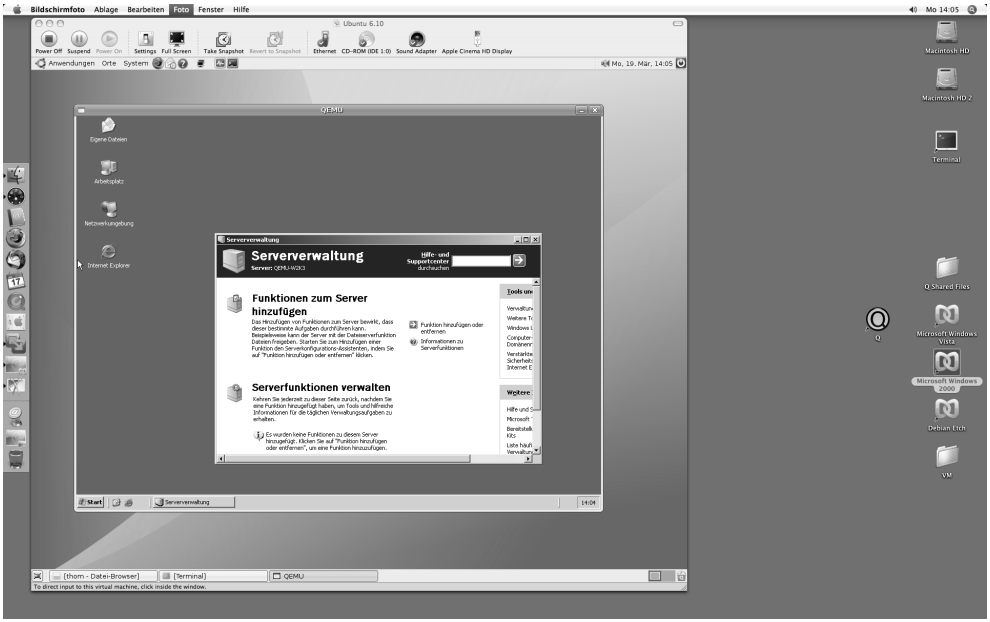

Abbildung 2: Unter Mac OS X läuft hier in VMware Fusion ein Ubuntu 6.10, in dem unter QEMU ein Microsoft Windows-2003-Server gestartet wurde.

QEMU lag zum Zeitpunkt der Drucklegung des Buchs in der Version 0.9.1 vor, noch nicht alle Ziele der Entwickler wurden komplett umgesetzt. QEMU erweist sich aber im Praxiseinsatz als stabil.

Für die x86-Architektur wird folgende Hardware emuliert:

- PC-BIOS vom Bochs-Projekt
- PC-Bus-Architektur: PCI und ISA-System (i440FX Host PCI Bridge und PIIX3 PCI to ISA Bridge)
- Symmetrisches Multiprozessorsystem (SMP) mit bis 255 CPUs
- Zwei PCI-ATA-Schnittstellen mit Unterstützung für maximal vier (virtuelle) Festplatten
- Grafikkarte (Cirrus CLGD 5446 PCI VGA-Karte oder Standard-VGA-Grafikkarte mit Bochs-VESA-BIOS-Extension)
- PS/2-Maus und -Tastatur
- CD-/DVD-Laufwerk
- Diskettenlaufwerk
- USB-Controller und virtueller USB-Hub
- Netzwerk (NE2000 PCI-Netzwerkadapter)
- Parallel-Port
- Serielle Ports
- Soundkarte (Soundblaster 16, ES1370 PCI)
- Eingebauter PC-Lautsprecher

Über dieses Buch

Dieses Buch ist zugleich ein Arbeitsbuch für Einsteiger und Fortgeschrittene und ein Nachschlagewerk. Es richtet sich an System-Administratoren, Softwareentwickler, Softwaretester und technisch Interessierte. Weitere Informationen findet man auch auf der Website zu diesem Buch ist *http://qemu-buch.de*.

In diesem Buch werden die folgenden Schreibweisen verwendet.

`Courierschrift`

Für Konsolen-Befehle und Quellcode.

`~# Befehl`

Für Unix-Konsolen-Befehle als User *root*.

`~$ Befehl`

Für Unix-Konsolen-Befehle als normaler User.

`Host ~# Befehl`

Für Unix-Konsolen-Befehle als User *root* auf dem Host-Rechner.

`Host ~$ Befehl`

Für Unix-Konsolen-Befehle als normaler User auf dem Host-Rechner.

`Host C:\> Befehl`

Für DOS/Windows-Konsolen-Befehle auf dem Host-Rechner. Da die QEMU-Optionen unabhängig vom Betriebssystem sind, wird hier meist die Schreibweise für die Unix-Shell verwendet.

`Gast ~# Befehl`

Für Unix-Konsolen-Befehle als User *root* auf dem Gast-System.

`Gast ~$ Befehl`

Für Unix-Konsolen-Befehle als normaler User auf dem Gast-System.

`Gast C:\> Befehl`

Für DOS/Windows-Konsolen-Befehle auf dem Gast-Rechner.

`(qemu) Befehl`

Befehle des QEMU-Monitors.

```
Host ~$ Ein sehr langer Befehl, \
der nicht in einer Zeile aufgelistet werden kann.
```

Wenn ein Befehl nicht in einer Zeile dargestellt werden kann, wird der Zeilenumbruch durch einen Backslash dargestellt. Der Befehl ist aber in einer Zeile einzugeben.

Grundlagen

Virtualisierung

Das primäre Ziel der Virtualisierung ist die Abkopplung der Software von den zur Verfügung stehenden Hardwareresourcen, um diese möglichst optimal bezüglich der Auslastung den einzelnen Systemen isoliert zuteilen zu können. Dabei geht es vor allem um die vier Kernkomponenten CPU, Hauptspeicher, Festplattenspeicher und Netzwerkanbindung.

Virtualisierung erlaubt so beispielsweise das Aufsetzen sehr spezieller virtueller Maschinen, die nur die für ihre Aufgabe nötigen Dienste enthält. So lassen sich zwar auf einer physikalischen Maschine sowohl ein Web-Server, ein Mail-Server und ein FTP-Server gleichzeitig betreiben, aber aus Sicherheitsgründen ist dies nicht empfehlenswert. Wird einer der drei Dienste kompromittiert, ist der komplette Server davon betroffen. Werden dagegen die drei Dienste in jeweils einer virtuellen Maschine voneinander isoliert auf dem System betrieben, muss nur das betroffene virtuelle System erneuert werden. Die anderen Dienste sind in ihrer virtuellen Umgebung nicht davon betroffen.

Als nützlicher Nebeneffekt kommt noch die Konsolidierung mehrerer einzelner Server auf einem entsprechend gut ausgestatteten System zum Tragen. Den virtuellen Maschinen werden genau die für eine bestimmte Aufgabe benötigten Ressourcen zugewiesen. Ändern sich die Anforderungen, werden die Ressourcen einfach darauf angepasst, ohne die Hardware verändern zu müssen. Ebenso leicht können überflüssige Ressourcen den virtuellen Maschinen auch wieder entzogen werden.

In der Welt der Virtualisierung unterscheidet man zur Zeit vier grundsätzliche Ansätze zum Betrieb fremder Systeme auf einer Plattform: die Hardware-Emulation, die Native Virtualization, die Paravirtualisierung und die Virtualisierung auf Betriebssystemebene.

Virtualisierungssoftware ist Software, die einen virtuellen Rechner bereitstellt, wobei die virtuell betriebenen Betriebssysteme Gast-Systeme heißen. Die Gast-Systeme werden von der eigentlichen Hardware, den vier Kernkomponenten des Hostsystems, isoliert.

Hardware-Emulation

Bei der Hardware-Emulation wird die komplette Hardware eines Computers simuliert und es wird auch die Installation und Ausführung von Betriebssystemen ermöglicht, die für andere Prozessoren konzipiert sind. Dazu wird unter anderem der komplette Befehlssatz des Gast-Prozessor mit Hilfe des Host-Prozessors nachgebildet. QEMU ist ein solcher Hardware-Emulator. Er erlaubt die Ausführung eines x86-Betriebssystems wie beispielsweise OS/2 auf einer anderen Architektur, wie zum Beispiel SPARC.

Die Emulation hat aber ihren Preis. Da jeder CPU-Befehl des Gast-Systems in einen entsprechenden Befehl (bzw. eine ganze Befehlsfolge) der CPU des Wirtssystems übersetzt werden muss, ist die Ausführungsgeschwindigkeit der Programme gering. Bei der Emulation älterer Hardware ist das zwar meist hinzunehmen, da bei der heutigen Hardware die Emulation oft schneller als die Originalhardware ist. Ein weiterer Nachteil ist aber das Fehlen einer dynamischen Ressourcenverwaltung – die Konfiguration der Systeme ist starr. Das heißt, während der Laufzeit können den Gast-Systemen die Ressourcen nicht variabel zugeteilt werden.

Native Virtualization

Die Native Virtualization, auch Hardware Virtualisierung genannt, stellt dem Gast-Betriebssystem Teilbereiche der physikalischen Hardware in Form von virtueller Hardware zur Verfügung. Damit werden allerdings nur Gäste bedient, die mit der CPU des Wirts kompatibel sind. Die privilegierten Befehle auf Ring 0 der CPU werden dabei vom Kernel des Wirts verwaltet. Entsprechende Anforderungen von den Gästen werden vom durch passende Ersatzfunktionen über den Kernel des Wirts abgewickelt. Zugriffe vom Gast auf Hardwaregeräte (Grafikkarte, Netzwerkkarte etc.) des Wirts werden über emulierte Standardkomponenten abgewickelt. Bekannte Vertreter dieser Virtualisierungsmethode sind beispielsweise die Produkte von VMware und Parallels. Die Ausführungsgeschwindigkeit der Gastsysteme ist bei dieser Methode sehr hoch.

Auch QEMU beherrscht mit dem optionalen Beschleuniger KQEMU (Seite 116) die Virtualisierung auf diese Art und erreicht zwar damit noch nicht ganz die Geschwindigkeit der VMware-Produkte, hat aber deutlich aufgeholt.

Paravirtualisierung

Im Gegensatz zur Hardware-Emulation und der Native Virtualization bietet die Paravirtualisierung nur ähnliche Schnittstellen wie die physikalische Hardware an. Die kritischen CPU-Befehle, die direkt die Hardware ansprechen, müssen vorher im Gast-System umgeschrieben worden sein und brauchen daher nicht mehr abgefangen und durch aufwendige Prozeduren ersetzt werden. Ein schlankes Programm, der Hypervisor, übernimmt die Scheduling-Aufgaben für die Ressourcen der virtuellen Maschinen. Der Nachteil ist, dass die Gast-Betriebssysteme angepasst werden müssen. Nur neuere Prozessoren (Intels Vanderpool, AMDs Pacifica) ermöglichen das Ausführen von unmodifizierten Gast-Systemen (hardwareunterstütze Virtualisierung). Ein Vorteil der Paravirtualisierung ist die wesentlich höhere Ausführungsgeschwindigkeit. Ein weiterer Vorteil ist die Live-Migration, bei der virtuelle Maschinen im laufenden Betrieb von einem Host-System auf ein anderes übertragen werden. Das ist zum Beispiel bei Wartungsarbeiten ein großer Vorteil. Die Gast-Systeme sind dann durchgehend verfügbar, auch wenn der Host heruntergefahren wird. Es wird zwischen zwei Hypervisor-Arten unterschieden:

Typ-1-Hypervisor: Der Hypervisor läuft direkt auf der Hardware. Die Gäste nutzen die Ressourcen, die vom Hypervisor zur Verfügung gestellt werden. Typisches Beispiel ist Xen (Open Source) und der ESX-Server von der Firma VMware, Inc., wobei der ESX-Server auch auf die Native Virtualization setzt.

Typ-2-Hypervisor: Der Hypervisor läuft auf einem Betriebssystem, das die I/O-Ressourcen bereitstellt. Typisches Beispiel ist KVM und der VMware-Server (ehemals GSX), wobei der VMware-Server auch mit Native Virtualization arbeitet.

Sowohl Xen als auch KVM verwenden übrigens Teile des Sourcecodes von QEMU für die Hardware-Emulation, wobei QEMU nicht dieser Art der Virtualisierung zugeordnet werden kann. Durch dessen gute Emulatorfähigkeiten ist aber Xen unter QEMU lauffähig (siehe Abschnitt „Hypervisor“, Seite 229).

Virtualisierung auf Betriebssystemebene

Bei der Virtualisierung auf Betriebssystemebene teilen sich mehrere Instanzen einen Kernel, weshalb sie auch Single Kernel Image (SKI) heißt. Hier wird also das Betriebssystem virtualisiert und nicht die Hardware. Das Host-Betriebssystem erzeugt weitere Instanzen seiner selbst. Aus diesem Grund werden nur Gast-Systeme mit dem gleichen Kernel wie das Host-System unterstützt. Da der Overhead gering ist, ist die Ausführungsgeschwindigkeit hoch. Eine Live-Migration der Gast-Systeme von einem Host auf einen anderen ist nicht möglich, weil der Kernel des Host-Systems nicht umziehen kann. Typische Vertreter dieser Kategorie sind die Solaris-Zones, BSD-Jails und Linux-VServer.

QEMU ist nicht zu dieser Art der Virtualisierung zuzuordnen. Es lassen sich aber Betriebssysteme, die mit dieser Virtualisierung arbeiten, unter QEMU betreiben (siehe Abschnitt Solaris, Seite 200).

Emulation

Im Zusammenhang mit Virtualisierung wird auch oft der Begriff Emulation genannt. Als Emulation wird in der Computertechnik das funktionelle Nachbilden eines Systems durch ein anderes bezeichnet. Das nachbildende System erhält die gleichen Daten, führt die gleichen Programme aus und erzielt die gleichen Ergebnisse wie das originale System. Ein Emulator ist also ein System, das ein anderes nachahmt. Es gibt viele Arten der Emulation. Es wurde bereits die Hardware-Emulation als eine Art der Virtualisierung beschrieben, bei der ein kompletter Computer emuliert wird. Es gibt aber auch Emulatoren, die nicht unter dem Begriff Virtualisierung einzuordnen sind. Dies ist zum Beispiel die API-Emulation.

API-Emulation

Bei dieser Art der Emulation werden die Schnittstellen und Bibliotheken eines anderen Betriebssystems (Application Programming Interface – API) nachgebildet. Das heißt, es lassen sich Programme eines anderen Betriebssystems verwenden. Dabei werden lediglich einige (bzw. alle) Funktionen des API (Applikation Programming Interface, Programmierschnittstelle) des Zielsystems auf dem Hostsystem emuliert. Diese Art der Emulation isoliert allerdings die ausgeführten Programme nicht vom System. Sie laufen samt Emulationsbibliotheken direkt im jeweiligen Hostsystem. Es handelt sich also nicht um eine Virtualisierung.

Typisches Beispiel ist Cygwin (siehe Seite 263). Cygwin stellt unter Windows einen großen Teil wichtiger API-Funktionen von Linux zur Verfügung. Zusätzlich gehören dazu eine Sammlung von Tools, wie sie unter Linux üblich sind. Unter Linux kompilierte Programme sind allerdings unter Cygwin nicht lauffähig, da nicht alle Bibliotheken und vor allem nicht alle Kernelfunktionen emuliert werden. Mit den auf Cygwin angepassten Werkzeugen (GCC-Compiler-Suite) und den mitgelieferten, angepassten Versionen wichtiger Bibliotheken, ist aber Erstellen und Ausführen vieler bekannter Programmpakete auch unter Windows möglich.

Der vielseitige QEMU unterstützt unter Linux und Mac OS X/Darwin auch eine OS-API-Emulation, die Userspace-Emulation (Seite 125). Anhand derer werden ausführbare Programme (Binaries), die für andere Bibliotheken kompiliert wurden, ausgeführt.

Vorteile von Virtualisierung und Emulation

Virtualisierung bringt eine Reihe von Vorteilen, die je nach Art der Virtualisierung unterschiedlich gewichtet sind. Dadurch, dass mehrere Gast-Systeme parallel auf einem Rechner laufen, ist eine bessere Hardware-Auslastung möglich. Oft werden, vor allem im Server-Umfeld, für wichtige Anwendungen jeweils ein physikalischer Computer zur Verfügung gestellt, damit es keine Wechselwirkung mit anderen und Beeinträchtigung durch andere Anwendungen gibt. Das Resultat ist ein größerer Rechnerpark mit entsprechend hohem Aufwand (Wartung, Stromversorgung, Klimaanlage). Betreibt man dagegen einzelne Anwendungen jeweils in einer virtuellen Maschine, sind die Anwendungen auch voneinander entkoppelt und die Hardware wird besser ausgenutzt. Neben der Sicherheit wird auch die Verfügbarkeit erhöht, denn virtuelle Maschinen können bei Hardwareproblemen auf eine andere Hardware umziehen und lassen sich komplett sichern und wiederherstellen.

Virtualisierung und Emulation ermöglichen Unabhängigkeit von der Hardware. Das heißt, Betriebssystem und Software werden von der Hardware losgelöst. So lassen sich zum Beispiel alte Versionen einer Anwendung parallel zur aktuellen Version auf neuer Hardware weiter betreiben oder Software für physikalisch nicht vorhandene Hardware entwickeln.

Virtuelle Maschinen können definierte Systemumgebungen auf unkomplizierte Weise zur Verfügung stellen. Mit Hilfe von Templates (virtuelle Maschinen mit fertig installierten Systemen) ist es zum Beispiel möglich, eine Maschine mit einer bestimmten Version eines Betriebssystems und einem gewünschten Patchlevel schnell auszurollen. So genannte Snapshots ermöglichen es, den Zustand einer virtuellen Maschine zu speichern, um später, zum Beispiel nach einem fehlgeschlagenen Patch, diesen Zustand wiederherzustellen. Dies ist besonders vorteilhaft für Entwicklungsumgebungen, Software-Tests, Support und Schulung. Virtuelle Maschinen sind auch ideal für die Bereitstellung von Honeypots; das sind Fallen für potentielle Angreifer auf ein System (zum Beispiel das Honeypot-System Argos, *http://www.few.vu.nl/argos*).

Installation von QEMU

Die Installation von QEMU ist für die gängigen Betriebssysteme unkompliziert und erfolgt jeweils mit den üblichen Methoden. Bei Betriebssystemen mit Paket- beziehungsweise Ports-Management wird das Paket beziehungsweise Port *qemu* installiert. Unter Microsoft Windows-NT-Versionen wird wie üblich das Setup-Programm heruntergeladen und gestartet. Die nachfolgenden Beispiele beziehen sich auf die QEMU-Version 0.9.1. Ältere Versionen besitzen nicht alle beziehungsweise teilweise andere Startoptionen. QEMU wird unter Linux entwickelt, weshalb es unter anderen Betriebssystemen daher noch einige Einschränkungen geben kann. Die Installationen von QEMU wurde von den Autoren auf unterschiedlicher Hardware durchgeführt. Dabei sind auf üblichen x86-PCs folgende BSD-, Linux- und Windows-Betriebssysteme im Einsatz:

- Pentium II, 233MHz, 392 MByte RAM, Microsoft Windows 2000, SP4; Knoppix 5.2
- AMD Athlon(tm) XP 2600+, 2 GHz, 0,5 GByte RAM, Ubuntu Linux
- Intel(R) Pentium(R) 4 CPU 3.00 GHz, 2 GByte RAM, Ubuntu Linux; PC-BSD
- Intel CPU T2400 1,83Ghz, 1 GByte RAM, Microsoft Windows XP; Microsoft Windows Vista Ultimate RC2

Für die Installation unter Mac OS X standen folgende Computer zur Verfügung:

MacBook Pro 15,4" ,2,33 GHz Intel Core 2 Duo, 2 GByte RAM, Mac OS X 10.4.9 mit aktuellem Sicherheits-Update (2007-004 Universal)

Mac Pro, 2 x 2,66 GHz Dual-Core Xeon, 6 GByte RAM, Mac OS X 10.4.9 mit aktuellem Sicherheits-Update (2007-004 Universal)

Die Installation auf Solaris erfolgte auf dieser Hardware:

SunBlade 150, UltraSPARC-IIe 650 Mhz, 2 GByte RAM, Solaris 10 11/06

QEMU wurde auch auf virtueller Hardware betrieben, so zum Beispiel unter QEMU und unter VMware und Parallels.

Auf allen beschriebenen Plattformen ist QEMU lauffähig. Bei der Hardware gilt die übliche Regel, dass je mehr RAM und CPU-Leistung zur Verfügung stehen, desto besser. Man kann kein System emulieren, das mehr Arbeitsspeicher benötigt, als das Host-System zur Verfügung stellt. Wer mehrere virtuelle Maschinen auf einen Host betreiben will, sollte bedenken, dass dazu ein schlankes, zuverlässiges System als Host-System nötig ist. Günstig sind die Ubuntu-Server- oder die Debian-Distribution. Die virtuellen Maschinen können von einem Host-System auf ein anderes mit anderer CPU, aber gleicher QEMU-Version übertragen werden. Zum Beispiel laufen auf Ubuntu Linux erzeugte virtuelle Maschinen mit Microsoft Windows NT oder OS/2 Warp 4 auf der SunBlade mit Solaris.

Microsoft Windows (NT, 2000, XP, Vista)

Man kann QEMU auf zwei Arten unter Microsoft Windows installieren. Die eigentlichen Komponenten von QEMU benötigen keine grafische Benutzeroberfläche (GUI). Wer ohne GUI auskommt, sollte die Variante ohne GUI bevorzugen. Sie eignet sich zum Beispiel dazu, eine DVD mit einer virtuellen Maschinen und der QEMU-Software zu brennen. Die virtuelle Maschine auf dieser DVD kann auf einem anderen Computer ohne Installation von QEMU direkt gestartet werden.

Wer lieber gleich ein Komplettpaket mit grafischer Benutzeroberfläche und Beschleuniger KQEMU möchte, installiert sich den Qemu-Manager für Windows. Man kann auch beide Versionen gleichzeitig installieren.

QEMU ohne GUI

Download: *http://www.qemu.org/download.html*

Die Installation von QEMU ohne GUI und ohne KQEMU unter den Microsoft Windows-NT-Versionen ist einfach und verändert nicht die Windows-Registry. Für diese Art der Installation werden auch keine Administratorrechte benötigt. Zuerst wird die neueste Version von QEMU für Windows heruntergeladen. Diese ZIP-Datei wird in ein Unterverzeichnis entpackt. Anschließend wechselt man in dieses Unterverzeichnis, in dem sich die Datei *qemu-win.bat* befindet. Beim Aufruf dieser Datei startet QEMU und bootet ein mitgeliefertes Mini-Linux von dem Image l*inux.img* in einem Fenster. Mit [Strg]+[Alt] wird dieses Fenster verlassen.

KQEMU

Download: *http://www.davereyn.co.uk/download.htm*

Bei der Installation des Qemu-Managers für Windows wird auch der optionale QEMU-Beschleuniger KQEMU installiert. Für eine separate Installation von KQEMU ist zuerst die Datei *kqemu-1.3.0pre11.tar.gz* zu entpacken. Danach ist in das angelegte Verzeichnis zu wechseln. Mit einem Rechtsklick auf die Datei *kqemu.inf* wird die Installation gestartet. Wie bei Microsoft Windows üblich, muss das Betriebssystem neu gestartet werden. Danach muss folgender Befehl eingegeben werden:

```
Host C:\> net start kqemu
```

Mit *net start* werden unter Microsoft Windows die Dienste gestartet. Weil dies nach jedem Neustart getan werden muss, kann diese Zeile in ein Startscript eingefügt werden.

Qemu-Manager für Windows

Download: *http://www.davereyn.co.uk/download.htm*

Der Qemu-Manager für Windows verwaltet die QEMU-Start-Optionen in einer einfach zu bedienenden grafischen Benutzeroberfläche. Die Installation ist einfach und selbsterklärend. Nach dem Herunterladen wird die Installation durch Anklicken der Datei *setupqemuk50.exe* aufgerufen. Ein Wizard hilft nach dem Start bei der Konfiguration. Seit der Version 4 des Qemu Managers für Windows kann auch Deutsch als Sprache eingestellt werden. Zuerst müssen der Pfad und die Version von QEMU angegeben werden. Danach sollte der KQEMU-Treibersupport aktiviert werden. Nun sind Pfade für die Floppy-, CD-, und DVD-Images sowie der Pfad für die virtuellen Festplatten zu konfigurieren, wonach der QEMU-Manager einsatzbereit ist. Der QEMU-Manager für Windows wird im Abschnitt „Grafische Benutzeroberflächen“ (Seite 131) genauer beschrieben.

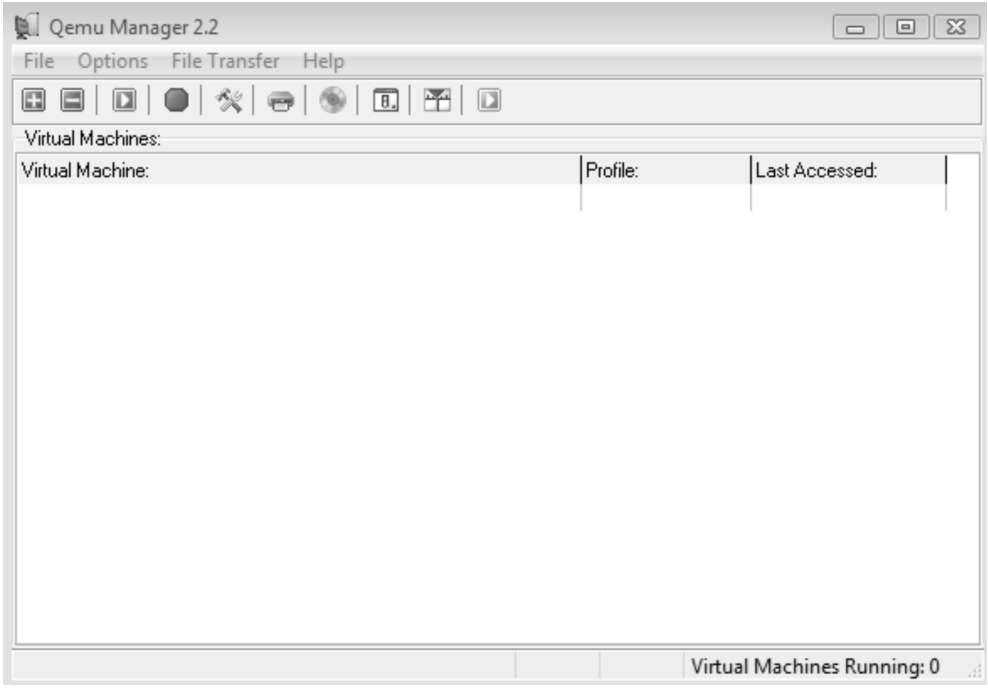

Abbildung 3: Der QEMU-Manager für Windows.

Linux

Vorkompilierte Binaries für Linux-i386

Download: *http://www.qemu.org/download.html*

Falls keine aktuellen QEMU-Pakete für die verwendete Linux-Distribution vorhanden sind, lädt man im Download-Bereich der QEMU-Website eine tar.gz-Datei mit x86-Binaries für Linux herunter, installiert diese durch einfaches Entpacken mit *tar* (Seite 272).

```
Host ~# tar xzvf qemu-0.9.1-i386.tar.gz -C /
```

Es ist auch möglich, QEMU ohne root-Rechte als normaler User zu installieren. Dazu entpackt man das tar-Archiv in ein Verzeichnis.

```
Host ~$ tar xzvf qemu-0.9.1-i386.tar.gz
```

Danach wechselt man in das von tar erstellte Unterverzeichnis. QEMU startet man mit den folgenden Befehl (Beispiel):

```
Host ~$ usr/local/bin/qemu -L usr/local/share/qemu/   \
-hda PFAD/PLATTE.img
```

Die Startoption *-hda* bindet die erste virtuelle Festplatte ein. Wichtig ist hier neben der Pfadangabe zu den QEMU-Binaries der Parameter *-L*, der den Pfad zum Verzeichnis mit der BIOS-Datei angibt. Es können auch die Inhalte der Verzeichnisse *usr/local/bin* und *usr/local/share/qemu* in ein Verzeichnis kopiert werden.

```
Host ~$ cp usr/local/bin/* .
Host ~$ cp usr/local/share/qemu/* .
```

Wird zum Beispiel eine virtuelle Maschine inklusive dieser QEMU-Dateien auf eine DVD gebrannt, kann diese auf einem anderen Rechner gestartet werden, ohne dass dort QEMU installiert sein muss. Der Aufruf vereinfacht sich dann wie folgt:

```
Host ~$ ./qemu -L . -hda PLATTE.img
```

Knoppix 5.2 (Live-DVD)

Website: *http://www.knoppix.org*

Bei Verwendung der Live-DVD Knoppix 5.2 ist keine Installation notwendig. Hier reicht es, den PC von dieser DVD zu booten. Allerdings benötigt eine Live-DVD viel Arbeitsspeicher, für die virtuellen Maschinen bleibt nicht viel übrig. Im Beispiel wird eine virtuelle Maschine mit einer im Host-System eingelegten DOS-Diskette gebootet. Dazu öffnet man eine Konsole und ruft QEMU auf.

```
Host ~$ qemu -boot a -fda /dev/fd0 -m 1
```

Debian 4.0 und Ubuntu 7.10

Die Installation von QEMU unter Debian und dessen Derivaten (Ubuntu) ist als User *root* mit einer Befehlszeile erledigt.

```
Host ~# apt-get install qemu
```

Es ist aber zu prüfen, ob QEMU in der Version 0.9.1 vorliegt. Die zu installierende Version kann mit einer simulierten Installation (*-s*) ermittelt werden.

```
Host ~# apt-get -s install qemu
```

Wer nicht gerne Befehle in der Konsole eingibt, kann auch die Installation mit dem grafischen Paketmanager *Synaptic* vornehmen. Zuvor sollten einmalig die entsprechenden Programmquellen aktiviert werden, was über das *Menü System | Administration | Software-Quellen* geschieht. Hier werden alle Quellen aktiviert. Die Synaptic-Paketverwaltung wird über das *Menü System | Administration | Synaptic-Paketverwaltung* aufgerufen. Nach Auswahl der Pakete wird die Installation aus dem Internet mit dem Button *Anwenden* gestartet.

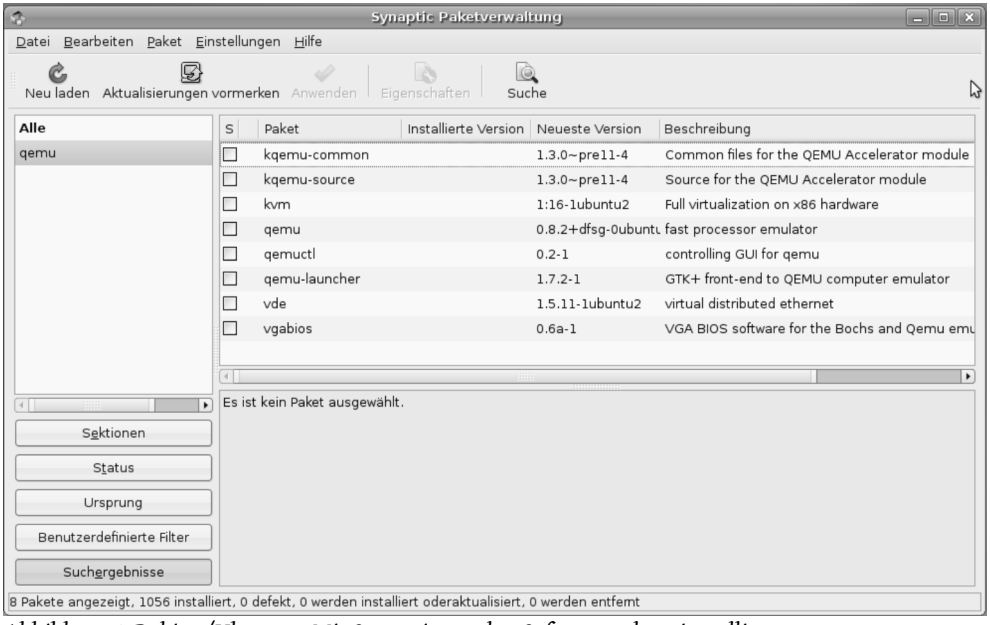

Abbildung 4: Debian/Ubuntu – Mit Synaptic werden Softwarepakete installiert.

OpenSuSE 10.3

Unter SuSE werden Konfigurationen und Softwareinstallationen mit dem Tool *YAST* vorgenommen. Im *YAST* geschieht das über den Punkt *Software installieren und löschen*. Zuvor werden die notwendigen Softwarequellen unter *Installationsquelle wechseln* hinzugefügt. Folgende URLs müssen hinzugefügt werden:

– *http://download.opensuse.org/distribution/10.3/repo/oss/*

– *http://download.opensuse.org/distribution/10.3/repo/non-oss/*

Jetzt kann mit *Software installieren und löschen* das Paket *qemu* gesucht und installiert werden.

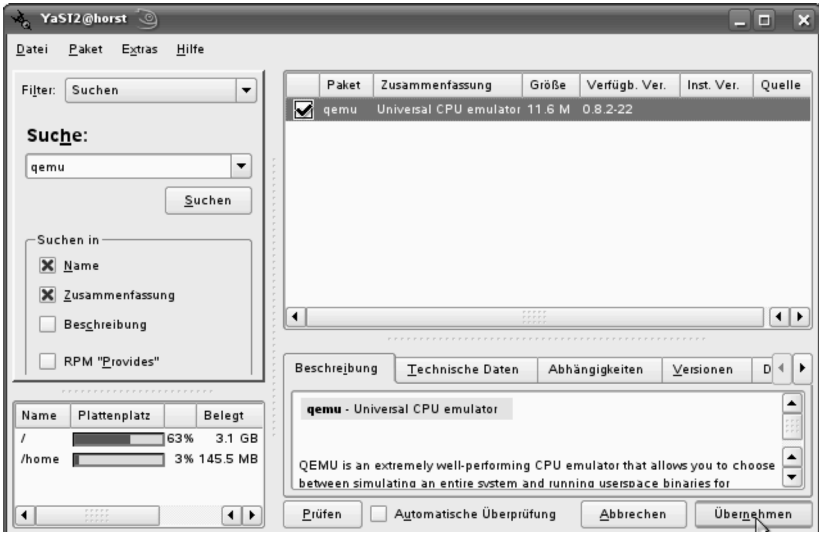

Abbildung 5: Software wird bei SuSE mit YAST installiert.

Win4Linux

Website: *http://www.win4lin.com*

Aufbauend auf QEMU gibt es mit Win4Lin ein kommerzielles Produkt, das speziell für eine einfache Bedienung, Installation und Konfiguration entwickelt wurde, mit dem alleinigen Ziel, Microsoft-Windows-Systeme innerhalb von Linux ausführen zu können. Die Installation wird in den mitgelieferten Handbüchern beschrieben.

BSD-Unix-Systeme

Mac OS X 10.4 (x86)

Download: *http://www.kju-app.org/kju*

Die folgenden Ausführungen beziehen sich auf aktuelle Apple-Hardware, basierend auf Intel-Prozessoren. Entsprechend der Apple-Philosophie, komplizierteste Systeme einfach bedienbar zu gestalten, wird QEMU im Paket mit einer funktionalen Benutzeroberfläche mit dem Namen *Q* bereitgestellt. Zu finden ist diese Software auf der oben angegebenen Website dieses Projektes. Es lassen sich zwei Versionen, nämlich die als stabil bezeichnete *Q-0.9.0a89* und eine aktuelle Entwicklerversion *Q-0.9.1d118* herunterladen. Weiterhin erhält man im Developer-Wiki Einblick in die Quelltexte (siehe Abschnitt „Quellen kompilieren“, Seite 34). Die einfachste und schnellste Installation geht mit folgenden Schritten:

– Herunterladen und Anhängen des Disk-Images *Q-0.9.1d118 .dmg*.
– Bestätigen der Lizenzbedingungen.
– Verschieben des Programmpakets *Q* in den Programmordner *Applications*.

Das war schon die komplette Installation. Der Emulator kann sofort gestartet werden, wie üblich per Doppelklick auf das Programmsymbol im Finder unter *Programme*. *Q* wird im Abschnitt „Grafische Benutzeroberflächen“ (Seite 151) genauer beschrieben.

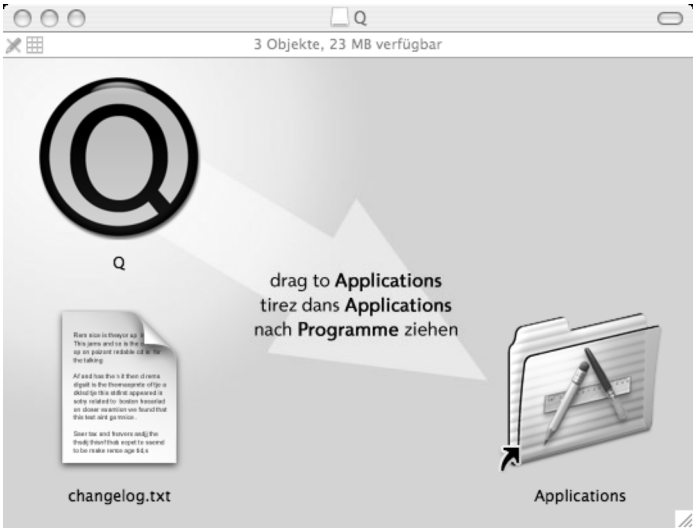

Abbildung 6: MacOS X mit geöffneten Disk-Image

FreeBSD 7 (x86, amd64)

Software wird für FreeBSD in Ports und Packages angeboten. Ports enthalten den Source-Code der Software und werden daher beim Installieren kompiliert. Bei der Installation von FreeBSD wird die Port Collection mit installiert, man kann diese durchsuchen. Zu QEMU relevante Ports befinden sich im Verzeichnis *emulator* im Port-Tree.

```
Host ~# cd /usr/ports/emulators/
```

In dem Unterverzeichnis *qemu* wird die Installation von QEMU gestartet.

```
Host ~# cd /usr/ports/emulators/qemu
Host ~# make install clean
```

Es ist auch möglich QEMU als Package zu installieren. *Package* sind bereits fertig kompilierte Software-Pakete. Oft sind aber Ports aktueller als Package. Hier die Installation von QEMU als Package:

```
Host ~# pkg_add -r qemu
```

Der optionale Beschleuniger KQEMU wird im Package mit folgendem Befehl installiert.

```
Host ~# pkg_add -r kqemu-kmod
```

KQEMU kann nun als Kernel-Modul geladen werden.

```
Host ~# kldload kqemu
Host ~# kldload aio
```

Damit nach jedem Neustart die Module *kqemu* und *aio* geladen werden, ist diese Zeile in das Start-Script */etc/rc.conf* einzutragen.

```
kqemu_enable="YES"
```

Es stehen auch die grafischen Benutzeroberflächen (siehe Abschnitt „Grafische Benutzeroberflächen“, Seite 131) *qemu-launcher* und *QtEmu* als Port und Package zur Verfügung. Hier als Beispiel die Installation von QtEmu als Package.

```
Host ~# pkg_add -r qtemu
```

Solaris 10 und OpenSolaris

Für die **x86/64-Architektur** steht ein aktuelles QEMU-Paket unter der Website *http://www.thoughtwave.com/downloads.html* unter dem Punkt *QEMU v0.9.1 with MTOOLS 3.9.10 for Solaris 10* zum Download bereit. Nach dem Download entpackt man diese Datei.

```
Host ~# bzip2 -d THOTqemu-0.9.1-20080113-universal-solaris10.bz2
```

Anschließend wird mit dem Tool *pkdadd* dieses Paket installiert.

```
Host ~# pkgadd -d THOTqemu-0.9.1-20080113-universal-solaris10
```

Es ist noch folgender Link zu setzen, damit die *Library libSDL-1.2.so.0* gefunden wird.

```
Host ~# ln -s /opt/thoughtwave/lib/libSDL-1.2.so.0   \
/lib/libSDL-1.2.so.0
```

Nach der Installation kann QEMU mit Angabe des Pfades aufgerufen werden.

```
Host ~$ /opt/thoughtwave/bin/qemu
```

Damit man nicht immer den kompletten Pfad angeben muss, kann der Suchpfad wie folgt erweitert werden.

```
Host ~$ export PATH=$PATH:/opt/thoughtwave/bin/
```

Der optionale Beschleuniger *KQEMU* ist aus den Quellen zu kompilieren. Dazu müssen erst die entsprechenden Kompilierungswerkzeuge installiert werden. Eine einfache Installation ist mit dem Tool *pkg-get* möglich. Auch dieses Tool muss erst installiert werden. Dazu lädt man es sich von der URL *http://www.blastwave.org/pkg-get.php* herunter und installiert es anschließend mit *pkgadd*.

```
Host ~# pkgadd -d pkg_get-3.7.2-all-CSW.pkg
```

Damit man nicht immer den kompletten Pfad zu *pkg-get* angeben muss, erweitert man den Suchpfad.

```
Host ~# export PATH=$PATH:/opt/csw/bin/
```

Als erstes ist die Paketliste zu aktualisieren.

```
Host ~# pkg-get -U
```

Nun können die notwendigen Kompilierungswerkzeuge installiert werden.

```
Host ~# pkg-get install gmake gcc4 gnustep_make wget
```

Jetzt wird das tar-Archiv mit dem Beschleunigermodul *KQEMU* von der Website *http://opensolaris.org/os/project/qemu/downloads/* heruntergeladen. Nach dem Herunterladen wird die Datei entpackt.

```
Host ~# bzip2 -d   \
kqemu-1.3.0pre11__sol10FCSplus_20070214_src_and_bins.tar.bz2
Host ~# tar xvf   \
kqemu-1.3.0pre11__sol10FCSplus_20070214_src_and_bins.tar
```

Man wechselt in das erstellte Verzeichnis und kompiliert die Quellen.

```
Host ~# cd kqemu-1.3.0pre11__osol20070214
Host ~# ./configure
Host ~# qmake
```

Es wird das Kernelmodul *kqemu-solaris-i386* angelegt. Dieses ist umzubenennen.

```
Host ~# mv kqemu-solaris-i386 kqemu
```

Mit folgenden Schritten wird das Kernelmodul installiert.

```
Host ~# /usr/sbin/install -f /usr/kernel/drv -m 755   \
-u root -g sys kqemu
new owner is root
kqemu installed as /usr/kernel/drv/kqemu
Host ~# /usr/sbin/install -f /usr/kernel/drv -m 644    \
-u root -g sys kqemu.conf
new owner is root
kqemu.conf installed as /usr/kernel/drv/kqemu.conf
```

Aktiviert wird das Modul mit *add_drv*.

```
Host ~# add_drv kqem
```

Zur Kontrolle, ob das Modul geladen wurde, kann der Befehl *modinfo* verwendet werden.

```
Host ~# modinfo | grep kqemu
215 f9c67000 1a608 207 1 kqemu (kqemu accelator v0.2)
```

Für die SPARC-Architektur steht ein aktuelles QEMU-Paket unter der Website *http://www.thoughtwave.com/downloads.html* unter dem Punkt *QEMU v0.9.1 with MTOOLS 3.9.10 for Solaris 10* zum Download bereit. Nach dem Download entpackt man die Datei.

```
Host ~# bzip2 -d THOTqemu-0.9.1-20080113-universal-solaris10.bz2
```

Anschließend wird mit dem Tool *pkdadd* dieses Paket installiert.

```
Host ~# pkgadd -d THOTqemu-0.9.1-20080113-universal-solaris10
```

Es ist noch folgender Link zu setzen, damit die *Library libSDL-1.2.so.0* gefunden wird.

```
Host ~# ln -s /opt/thoughtwave/lib/libSDL-1.2.so.0   \
/lib/libSDL-1.2.so.0
```

Nach der Installation kann QEMU mit Angabe des Pfades aufgerufen werden.

```
Host ~$ /opt/thoughtwave/bin/qemu
```

Damit man nicht immer den kompletten Pfad angeben muss, kann der Suchpfad wie folgt erweitert werden.

```
Host ~$ export PATH=$PATH:/opt/thoughtwave/bin/
```

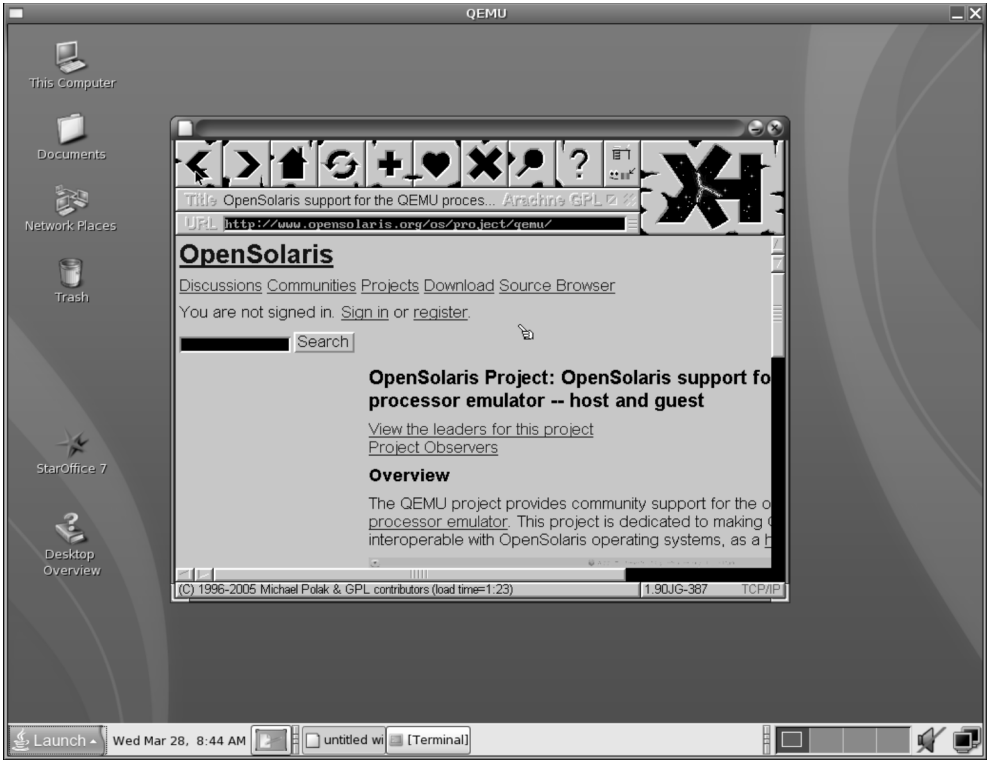

Abbildung 7: Unter Solaris 10 läuft hier FreeDOS in QEMU, wobei Solaris 10 selbst in QEMU gestartet wurde.

Win4BSD

Download: *ftp://ftp.win4bsd.com/pub/* (Testversion)

Aufbauend auf QEMU gibt es mit Win4BSD (*http://www.win4bsd.com/content/*) ein kommerzielles Produkt, das eine virtuelle Maschine zur Verfügung stellt, in der sich Microsoft Windows 2000 oder Microsoft Windows XP installieren und benutzen lässt . Die Installation wird in den mitgelieferten Handbüchern beschrieben.

Quellen kompilieren

Das Kompilieren der Quellen ist oft nicht erforderlich, da QEMU für die gebräuchlichsten Linux-Distributionen und Unix-Versionen als Softwarepaket zur Verfügung steht. Unter Microsoft Windows ist das Kompilieren aus Quellen unüblich. Diese Anleitung ist für Besitzer von Betriebssystemen gedacht, für die QEMU nicht als installierbares Paket vorliegt oder für solche Anwender, die QEMU modifizieren möchten oder eine aktuellere Version, als die in der Distribution gepflegte, einsetzen möchten. Die dazu notwendigen Shell-Befehle werden im Anhang unter Tools erläutert (Seite 262). Die Quellen werden von der QEMU-Website heruntergeladen und kompiliert.

```
Host ~# tar xzvf qemu-0.9.1.tar.gz
Host ~# cd qemu-0.9.1
Host ~# ./configure
Host ~# make
Host ~# make install
```

Für den optionalen QEMU-Beschleuniger KQEMU findet man auf der QEMU-Website die neueste Version unter dem Punkt *QEMU Accelerator Module*. Die Quellen von KQEMU sind wie üblich zu kompilieren. Es ist zuerst das tar-Archiv zu entpacken.

```
Host ~# tar xzvf kqemu-1.3.0pre11.tar.gz
Host ~# cd kqemu-1.3.0pre11
```

Kompiliert wird als User *root*.

```
Host ~# ./configure
Host ~# make
Host ~# make install
```

Das kompilierte Kernel-Modul wird mit dem Befehl *modprobe* geladen.

```
Host ~# modprobe kqemu
```

Dabei wird das Device */dev/kqemu* angelegt.

```
Host ~# ls -l /dev/kqemu
crw-rw-rw- 1 root root 250, 0 2006-09-17 14:24 /dev/kqemu
```

Über diese Gerätedatei kommunizieren dann die vom Benutzer angelegten und ausgeführten virtuellen Maschinen mit dem Kernelmodul *kqemu*. KQEMU ist jetzt einsatzbereit. Damit das Modul beim Systemstart des Hosts automatisch geladen wird, kann bei Debian-Derivaten eine Zeile mit *KQEMU* in der Datei */etc/modules* eingetragen werden. Es kann auch der Befehl *modprobe kqemu* in ein Bootscript geschrieben werden.

Neuere Linux-Distributionen mit Kernel 2.6 benutzen anstelle des betagten *devfs* zur Verwaltung der Kernel-Gerätedateien das neu entwickelte *udev*. Damit ist es möglich, auch Hotplug-Geräte zur Laufzeit automatisch ins System einzubinden. Die Konfiguration von

udev erfolgt über Konfigurationsdateien in */etc/udev*. Wenn die Linux-Distribution *udev* unterstützt, wie dies bei Ubuntu und Fedora der Fall ist, wird das Device */dev/kqemu* nicht automatisch beim Booten angelegt. Dazu sind in ein Bootscript folgende Zeilen einzutragen.

```
mknod /dev/kqemu c 250 0
chmod 666 /dev/kqemu
```

Der sauberere Weg besteht allerdings im Eintragen des Kernelmoduls in die Konfiguration von *udev*, wie es exemplarisch im Anschluss für die Installation von QEMU/KQEMU unter Ubuntu/Debian beschrieben wird.

Ubuntu 6.10 (x86) und Debian 4.0 (x86)

Download: *http://www.qemu.org/download.html*

Für das Kompilieren der Quelltexte unter Ubuntu 6.10 und Debian 4.0 (Etch) sollten die Quellen im Home-Verzeichnis unter *source* abgelegt werden. Kompiliert wird die Software mit normalen Benutzerrechten, erst die Installation in die Zielverzeichnisse erfordert root-Rechte. Zunächst werden die Quellpakete für QEMU (*qemu-0.9.1.tar.gz*) und KQEMU (*kqemu-1.3.0pre11.tar.gz*) von der QEMU-Website heruntergeladen und entpackt.

```
Host ~$ mkdir source
Host ~$ cd source
Host ~$ wget http://www.qemu.org/qemu-0.9.1.tar.gz
Host ~$ wget http://www.qemu.org/kqemu-1.3.0pre11.tar.gz
Host ~$ tar xzf qemu-0.9.1.tar.gz
Host ~$ tar xzf kqemu-1.3.0pre11.tar.gz
```

Für das Kompilieren von QEMU unter Ubuntu 6.10 sind zwei weitere Pakete zu installieren. Der Standard-Compiler unter Ubuntu ist *gcc-4.1*, mit ihm kommt QEMU nicht zurecht. Aus diesem Grund ist der Compiler *gcc-3.3* zusätzlich zu installieren.

```
Host ~$ sudo apt-get install gcc-3.3
```

Ebenfalls benötigt wird das Entwicklerpaket *libsdl1.2-dev*.

```
Host ~$ sudo apt-get install libsdl1.2-dev
```

Da *apt-get* automatisch die Paketabhängigkeiten prüft, werden mit dieser Aktion einige Megabyte an Entwicklerpaketen zusätzlich installiert. Danach kann die Kompilierung von QEMU beginnen. Dazu sollte der symbolische Link */usr/bin/gcc* durch einen Link auf *gcc-3.3* ersetzt werden. Das configure-Script erkennt zwar die richtige Compiler-Version selbständig, kommt aber zu keinem Ergebnis.

```
Host ~$ sudo ln -sf /usr/bin/gcc /usr/bin/gcc-3.3
```

Nach dem Wechsel ins Quellverzeichnis wird zunächst das *Makefile* konfiguriert und der Übersetzungsvorgang gestartet. Anschließend werden die fertigen Programmdateien installiert.

```
Host ~$ cd /home/USER/source/qemu-0.9.1
Host ~$ ./configure -enable-linux-user
Host ~$ make
Host ~$ sudo make install
```

Damit ist QEMU bereits benutzbar. Der optionale Beschleuniger KQEMU ist noch zu übersetzen und ins System einzubinden. Dazu ist wieder auf Version 4.1 des Compilers zu verweisen, ins Quellverzeichnis zu wechseln und das *Makefile* zu konfigurieren. Dann ist der Übersetzungsvorgang anzustoßen.

```
Host ~$ sudo ln -sf /usr/bin/gcc /usr/bin/gcc-4.1
Host ~$ cd /home/USER/source/kqemu-1.3.0pre11
Host ~$ ./configure
Host ~$ make
```

Jetzt ist das Kernelmodul zu installieren.

```
Host ~$ sudo make install
```

Das fertige Kernelmodul wird mit dem Befehl *modprobe* geladen.

```
Host ~$ sudo modprobe kqemu
```

Mit dem Laden des Moduls wird automatisch das Device */dev/kqemu* von *udev* angelegt. Allerdings sind die Zugriffsrechte darauf sehr restriktiv, lediglich der User *root* darf dieses Device nutzen. Um normalen Benutzern den Zugriff darauf zu gewähren, müssen entsprechende Rechte zugewiesen werden.

```
Host ~$ sudo chmod 666 /dev/kqemu
```

Das Laden des Moduls und die Zuweisung der Rechte an das Device */dev/kqemu* soll automatisch beim Systemstart erfolgen. Dazu ist die Datei */etc/rc.local* mit der folgenden Zeile zu ergänzen.

```
modprobe kqemu
```

Ebenfalls in dieser Datei muss ein Eintrag zur Einstellung einer höheren Auflösung der Real-Time-Clock, der von QEMU beim Start gewünscht wird, ergänzt werden.

```
echo 1024 > /proc/sys/dev/rtc/max-user-freq
```

Damit wird die Frequenz der RTC auf 1024 Hz gesetzt.

Angelegt wird die Gerätedatei für das Kernelmodul *kqemu* automatisch von *udev*. Konfiguriert wird *udev* über die Dateien im Verzeichnis */etc/udev/rules.d*. Die Rechte der Geräte werden in der Datei *40-permissions.rules* verwaltet. Darin ist folgende Zeile anzufügen:

```
KERNEL=="kqemu", MODE="0666"
```

Beim nächsten Start wird das Kernelmodul automatisch geladen und die Rechte für das zugehörige Device werden so gesetzt, dass alle Benutzer lesend und schreibend auf diese Gerätedatei zugreifen können.

Zum Kompilieren der Software unter **Debian 4.0** (Etch) sind einige zusätzliche Pakete zu installieren. Die GNU C Compilersuiten *gcc-3.4* und *gcc-4.1* werden als Werkzeug für die Übersetzung benötigt. Weiterhin werden die Headerdateien der Entwicklerversionen der Bibliotheken *libsdl1.2-dev* und *zlib1g-dev* und ein kleines Hilfsprogramm zum Erstellen von Kernelmodulen (*modpost*) gebraucht. Dazu sind die kompletten Quellen des Kernelzweigs der Distribution (*linux-tree-2.6.18*) und das Paket *kernel-package* einzuspielen. Diese Pakete können über die Synaptic-Paketverwaltung (*Menü System | Systemverwaltung | Synaptic-Paketverwaltung*) installiert werden. Schneller und einfacher geht es aber mit Shell-Befehlen. Für die Installation sind zunächst root-Rechte nötig.

```
Host ~$ su -
Password: *****
```

Jetzt können die einzelnen Pakete mit *apt-get* ins System eingespielt werden.

```
Host ~# apt-get install gcc-3.4 gcc-4.1
Host ~# apt-get install libsdl1.2-dev zlib1g-dev
Host ~# apt-get install linux-tree-2.6.18 kernel-package
```

Die Quellen des Kernels landen dabei als tar-Archiv (*linux-source-2.6.18.tar*) im Verzeichnis */usr/src*. Das tar-Archiv muss entpackt werden.

```
Host ~# cd /usr/src
Host ~# tar xf linux-source-2.6.18.tar
```

Aus Gründen der Bequemlichkeit sollte man direkt einen symbolischen Link vom Quellverzeichnis */usr/src/linux-source-2.6.18* auf */usr/src/linux* anlegen.

```
Host ~# ln -s /usr/src/linux-source-2.6.18 /usr/src/linux
```

Jetzt muss noch die aktuelle Konfiguration des Kernels in das Quellverzeichnis kopiert werden. Zu finden ist die Konfiguration im Verzeichnis */boot*.

```
Host ~# cp /boot/config-2.6.18-4-686 /usr/src/linux/.config
```

Damit jeweils die richtige Version des Compilers (*gcc-3.4* oder *gcc-4.1*) verwendet wird, ist ein symbolischer Link auf die gerade benötigte Version anzulegen. Für die Übersetzung des Kernels wird Version 4.1 gebraucht.

```
Host ~# ln -sf /usr/bin/gcc-4.1 /usr/bin/gcc
```

Damit sind die Voraussetzungen zum Neubau des Kernels erfüllt. Ein neuer Kernel wird zwar nicht benötigt, ein wichtiges Hilfsprogramm zur Erzeugung von Kernelmodulen (*modpost*) wird dabei automatisch angelegt. Bis zur Kernelversion 2.6.13.4 ging das einfacher, in der aktuellen Version ist aber dieser Umweg nötig.

```
Host ~# make-kpkg configure
Host ~# make-kpkg kernel_image
```

Mit *make-kpkg* lassen sich Debian-Kernelpakete aus dem Kernelquellcode erzeugen. Dabei kann sowohl die Konfiguration des Kernels während des Einsatzes von *make-kpkg* vorgenommen werden, als auch der eigentliche Übersetzungsvorgang (Kompilieren) des Quellcodes angestoßen werden. *make-kpkg* muss immer in dem Verzeichnis aufgerufen werden, in dem sich der Quellcode des Kernels befindet. Dazu sind administrative Rechte notwendig. Dann kann QEMU übersetzt werden. Dazu ist zunächst der Compiler *gcc* in der Version 3.4 zu aktivieren, weil QEMU nicht von den Compilern der 4.x-Reihe übersetzt werden kann.

```
Host ~# ln -sf /usr/bin/gcc-3.4 /usr/bin/gcc
```

Jetzt werden die Makefiles für QEMU konfiguriert und QEMU mit einfachen Benutzerrechten übersetzt. Erst zum Installieren werden wieder root-Rechte benötigt.

```
Host ~$ cd /tmp/qemu-0.9.1
Host ~$ ./configure ––enable-linux-user
Host ~$ make
Host ~$ su -
Password: ****
Host ~# make install
```

Die Programmdateien von QEMU liegen unter */usr/local/bin* zur Benutzung bereit. Als nächstes wird das Kernelmodul *kqemu* kompiliert. Dies erfordert wieder den neueren Compiler.

```
Host ~# ln -sf /usr/bin/gcc-4.1 /usr/bin/gcc
```

Das Konfigurieren der Makefiles und das Übersetzen des Moduls geschieht mit normalen Benutzer-Rechten, für das Installieren sind root-Rechte notwendig.

```
Host ~$ cd /tmp/kqemu-1.3.0pre11
Host ~$ ./configure
Host ~$ make
Host ~$ su -
Password: *****
Host ~# make install
```

Das fertige Kernelmodul wird mit dem Befehl *modprobe* geladen.

```
Host ~# modprobe kqemu
```

Mit dem Laden des Moduls wird automatisch das Device */dev/kqemu* von *udev* angelegt. Allerdings sind die Zugriffsrechte sehr restriktiv, lediglich der User *root* darf dieses Device nutzen. Um normalen Benutzern den Zugriff zu gewähren, müssen noch entsprechende Rechte definiert werden.

```
Host ~# chmod 666 /dev/kqemu
```

Das Laden des Moduls und die Zuweisung der Rechte an das Device */dev/kqemu* sollen automatisch beim Systemstart erfolgen. Dazu ist die Datei */etc/rc.local* mit der folgenden Zeile zu ergänzen.

```
modprobe kqemu
```

Angelegt wird das Device */dev/kqemu* dabei automatisch von *udev*. Konfiguriert wird *udev* über Dateien im Verzeichnis */etc/udev*. Die Rechte der Geräte werden in der Datei *permissions.rules* verwaltet. Darin ist folgende Zeile anzufügen:

```
KERNEL=="kqemu", MODE="0666"
```

Beim nächsten Start wird das Kernelmodul automatisch geladen und die Rechte für das zugehörige Device werden so gesetzt, dass alle Benutzer lesend und schreibend auf diese Gerätedatei zugreifen können.

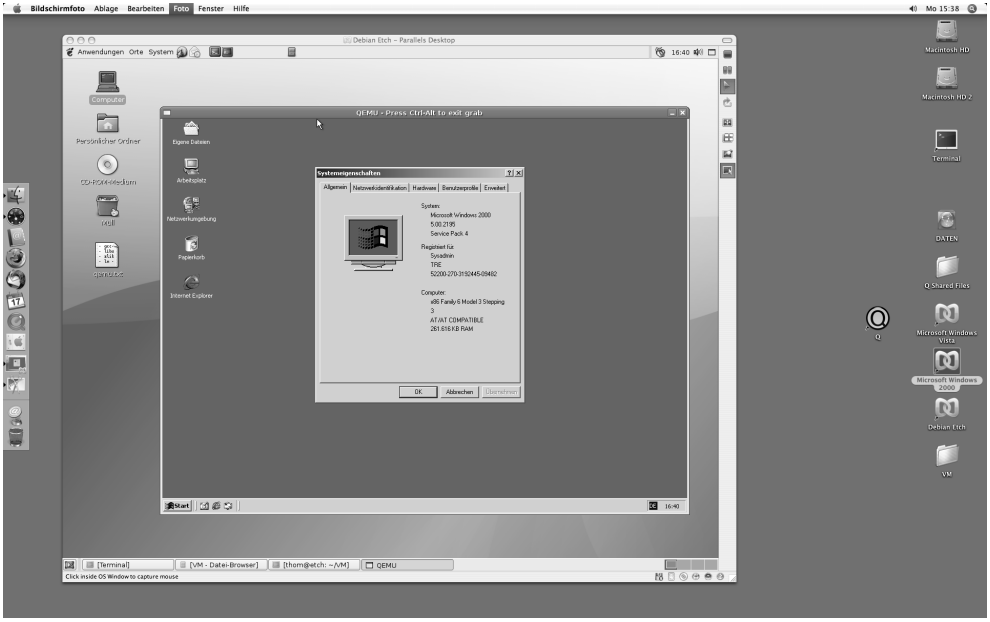

Abbildung 8: Auch die Installation von QEMU kann unter QEMU getestet werden. Unter Mac OS X wird hier mit Paralels Desktop ein PC für Debian 4.0 (Etch) emuliert. Darin virtualisiert QEMU eine Maschine für Windows 2000.

Mac OS X 10.4 (x86)

Unter der Mac-Benutzeroberfläche werkelt ein Unix mit BSD-Wurzeln. Apple hat zwar an einigen Stellen kräftig umgebaut, aber die Anpassung vieler Open-Source-Projekte an Mac OS X ist trotzdem relativ einfach möglich. Als Compiler nutzt Apple beispielsweise die gcc-Suite. Die Entwicklungsumgebung *Xcode*, die alle benötigten Ressourcen zum Erzeugen eigener Programme enthält, gehört zum Lieferumfang von Mac OS X. Sie ist allerdings nicht standardmäßig installiert und muss ins System übernommen werden. Zu installieren ist dazu das Paket *XcodeTools.mpkg*, das sich auf der Mac OS X *Install Disc 1* im Ordner *Xcode Tools* befindet.

Die Quellen von QEMU werden direkt aus dem Versionskontrollsystem CVS geladen, der dazu benötigte Client ist bereits im System enthalten. Die Anpassungen samt Benutzeroberfläche für den Mac werden aus einem Subversion-Repository bezogen. Der Client ist noch nicht im System vorhanden und muss daher installiert werden. Fertig angepasste Pakete des Subversion-Clients bezieht man über das Fink-Projekt. *Fink* ist eine Portierung der komfortablen Paket-Management Tools *dpkg* und *apt-get* aus der Debian-Welt. Neben den Befehlszeilenwerkzeugen ist mit dem *Fink Commander* auch eine grafische Benutzeroberfläche im Paket enthalten.

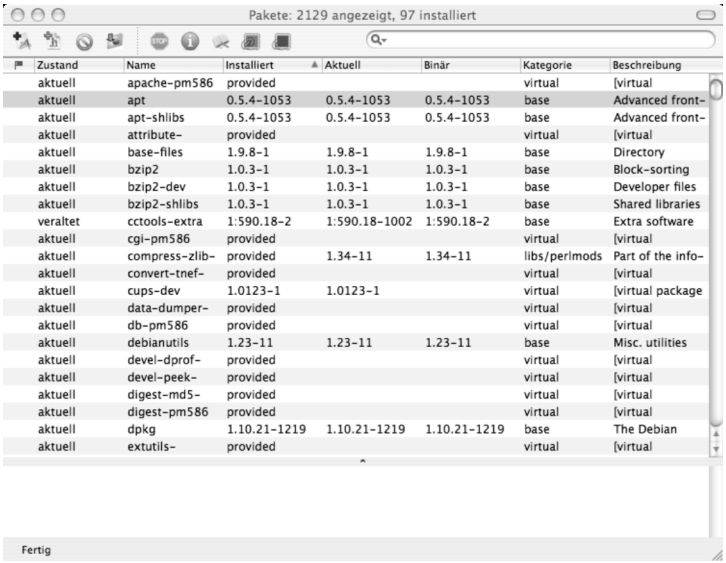

Abbildung 9: Mac OS X – Der Fink-Commander.

Bezogen werden kann Fink über die Projekt-URL *http://fink.sourceforge.net*. Das Disk-Image enthält das Installationspaket *Fink 0.8.1-Intel Installer.pkg* und die Grafikoberfläche im Ordner *Fink-Commander*. Das Installationspaket wird einfach ausgeführt. Die Programme landen dabei im neu angelegten Verzeichnisbaum */sw*. Die Shellvariable *$PATH* wird automatisch um */sw/bin* und */sw/sbin* ergänzt. Damit werden die per Fink nachinstallierten ausführbaren Programme auch automatisch vom System gefunden. Danach ist der Subversion-Client *svn-client* (Version 1.2.3-1012) zu installieren. Dies ist über den Fink-Commander mit Mausunterstützung möglich, die wesentlich schnellere Lösung ist die Eingabe des folgenden Befehls:

```
Host ~$ sudo apt-get install svn-client
```

Damit sind alle Vorbereitungen zum Selbstbau von *Q* getroffen. Jetzt fehlen noch die nötigen Quellen. Sie werden direkt aus den jeweiligen Versionsverwaltungen bezogen. Für die Version *Q-0.9.0.d84* (Stand 06.05.2007) lässt sich dazu folgendes Script anwenden:

```
#!/bin/sh
rm -r -f products
rm -r -f tmp
#get/update libtransmission
svn co -r 480 svn://svn.m0k.org/Transmission/trunk/ Transmission
cp Transmission/mk/lib.mk Transmission/libtransmission/Makefile
#get/update QEMU
export CVS_RSH="ssh"
cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/qemu
co -D "2007-02-05 23:59" qemu
# get/update [kju:]
svn co http://www.kju-app.org/svn/q/trunk q
mkdir products
mkdir products/i386
mkdir products/ppc
mkdir products/universal
mkdir tmp
```

Eine aktuelle Version dieses Scripts (*prepare.sh*) ist über die Projektsite *http://www.kju-app.org/proj/wiki/DevelopmentOfKju* verfügbar. Zu beachten ist dabei besonders der ausgewiesene Versionsstand von QEMU. In diesem Beispiel ist es *2007-02-05 23:59*. Hier wird immer ein definierter Stand benutzt, während die Mac-Anpassung *Q* aus dem aktuellen Zweig des Repositories stammt. Dieser kann sich täglich ändern. Eventuelle Änderungen am Stand des QEMU-Parts sollten penibel beachtet werden, was sich durch das jeweils aktuelle Prepare-Script realisieren lässt. Das Script wird in ein beliebiges Verzeichnis (zum Beispiel */PDB/daten/sources/qemu*) heruntergeladen und ausführbar gemacht.

```
Host ~# cd /PDB/daten/sources/qemu
Host ~# chmod u+x preprare.sh
```

Vom Script wird die nötige Verzeichnisstruktur erzeugt und die Quelldateien werden aus den Repositories der Versionsverwaltung heruntergeladen. Die zum Build der Software benötigten Scripte befinden sich im Verzeichnis *q*. Auch sie müssen noch ausführbar gemacht werden.

```
Host ~# cd q
Host ~# chmod u+x ./*.sh
```

Jetzt sind die Vorbereitungen zum Erzeugen des Programmpakets abgeschlossen, nun ist das passende Script im Verzeichnis *q* auszuführen.

```
Host ~# ./build_i386.sh
```

Nachdem der Build-Prozess ohne Fehler abgeschlossen ist (Warnungen können ignoriert werden), befindet sich das fertige Programmpaket im Ordner *products/i386*. Von hier aus wird es mit dem Finder ins Programmverzeichnis verschoben. Der Finder zeigt dabei lediglich das Programmsymbol, hinter dem sich allerdings die komplette Verzeichnisstruktur des Pakets verbirgt. Interessenten können sich im Finder den Paketinhalt anzeigen lassen.

Quickstart

Damit Ungeduldige nicht erst das Buch komplett durchlesen müssen, hier eine kurze Einführung für das Anlegen einer virtuellen Maschine unter QEMU. Dabei werden folgende Schritte ausgeführt: Zuerst wird eine virtuelle Festplatte (Image-Datei) angelegt. Dann wird QEMU mit dieser virtuellen Festplatte und dem Installations-Medium, zum Beispiel mit einer Installations-CD, gestartet. Die virtuelle Maschine wird mit den Startoptionen von QEMU konfiguriert. Das Betriebssystem wird, wie bei einer realen Maschine, installiert. Dabei ist nur auf die Unterstützung der von QEMU emulierten Hardware zu achten. Spezielle Treiber müssen im Gast-System nicht installiert werden.

Kommandozeile (alle Betriebssysteme)

Konfiguriert werden die virtuellen Maschinen unter QEMU mit seinen Startoptionen. Zur Demonstration der wichtigsten Optionen wird im nachfolgenden Beispiel eine virtuelle Maschine mit Shell-Befehlen erzeugt. Die Startoptionen von QEMU sind unter allen Betriebssystemen (Linux, Unix, Mac OS X, Microsoft Windows) gleich. Der Aufruf von QEMU auf der Kommandozeile hat auch den Vorteil, dass mögliche Fehlermeldungen sofort angezeigt werden.

Um dieses Quickstart-Beispiel möglichst einfach zu halten und schnell zu einem Ergebnis zu kommen, wurde das Betriebssystem Haiku (*http://www.haiku-os.org*) als Gastsystem ausgewählt. Haiku ist ein Open-Source-Projekt mit dem Ziel das einfach zu bedienende Betriebssystem BeOS nachzuprogrammieren und zu erweitern (siehe Seite 226). Das Anlegen der virtuellen Festplatte und die Installation des Betriebssystems ist hier nicht notwendig, da täglich ein aktualisierte Festplatten-Image mit bereits installiertem Betriebssystem von der URL *http://sikosis.com/haiku/haiku.image.bz2* heruntergeladen werden können. Nach dem Herunterladen muss das komprimierte Image (Raw-HDD) noch entpackt werden. Unter Windows geht das mit 7-Zip und unter Linux mit p7zip (siehe Seite 273). Nach dem Entpacken ist das Image in QEMU direkt nutzbar. Dazu wird *qemu* mit der Option *-hda* gefolgt von dem Namen der Image-Datei gestartet.

```
Host ~$ qemu -hda haiku.image
```

Hinweis: Sollte beim Start die Fehlermeldung erscheinen, dass die Datei *bios.bin* nicht gefunden wurde, muss zusätzlich noch mit der QEMU-Startoption *-L* der Pfad zu dieser Datei angegeben werden. Befindet sich die Datei *bios.bin* im aktuellen Pfad, ist ein Punkt zu verwenden. Dies ist der Fall, wenn QEMU nicht mit Administrationsrechten installiert wurde (siehe im Abschnitt „Installation von QEMU“ unter Windows – QEMU ohne GUI, Seite 24). Die Datei *bios.bin* enthält das PC-BIOS. Genau wie ein realer Rechner, startet auch der virtuelle PC mit den im BIOS abgelegten Routinen. Damit wird die virtuelle Hardware konfiguriert und definierte Schnittstellen dem zu installierenden bzw. zu startenden Betriebssystem zur Verfügung gestellt.

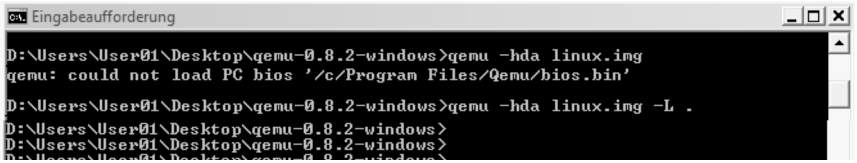

Abbildung 10: Unter Umständen muss beim Start von QEMU auf das Verzeichnis mit der Datei bios.bin verwiesen werden.

`Host C:\QEMU>` **`qemu.exe -hda haiku.image -L .`**

Sind alle Startoptionen von QEMU richtig, startet der virtuelle PC in einem Fenster.

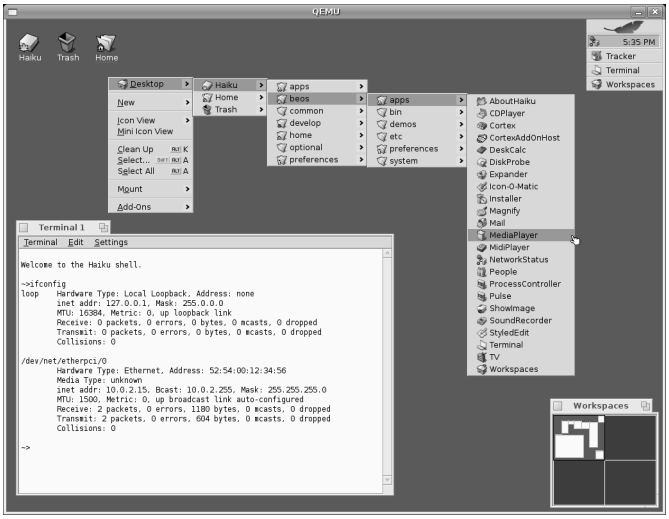

Abbildung 11: QEMU mit dem Betriebssystem Haiku.

Durch Klicken in das QEMU-Fenster werden die Maus- und Tastatur-Eingaben an das Gast-System übergeben. Das QEMU-Fenster wird mit [Strg]+[Alt] verlassen, zurück gelangt man wieder durch einen Klick in das QEMU-Fenster.

QEMU konfiguriert per integriertem DHCP-Server die Netzwerkeinstellungen des Gast-Systems. Das heißt, die eigene IP-Adresse, die Netzmaske, die IP-Adresse des Gateways und die IP-Adresse des DNS-Servers werden automatisch an das Gastsystem übergeben.

Qemu-Manager für Windows

Nachfolgend werden die Schritte zum Einrichten einer virtuellen Maschine am Beispiel ReactOS (*http://www.reactos.org/de/*) als Gast-Betriebssystem mit dem Qemu-Manager für Windows beschrieben. ReactOS ist Open-Source und benötigt wenig Ressourcen. Die Installations-CD ist schnell aus dem Internet heruntergeladen. Die Installation von ReactOS ähnelt der von Microsoft Windows XP, da ReactOS kompatibel zu XP-Betriebssystemen werden soll. Derzeit befindet sich ReactOS noch in der Entwicklung.

Zuerst lädt man das Image der Installations-CD von der Website herunter und entpackt es in das media-Verzeichnis des Qemu-Managers, das sich unter *C:\Programme\QemuManager\media* befindet. Unter Windows Vista benötigt man dazu Administratorrechte. Der Aufruf des Qemu-Managers erfolgt über das *Start-Menü | (Alle) Programme | Qemu-Manager*.

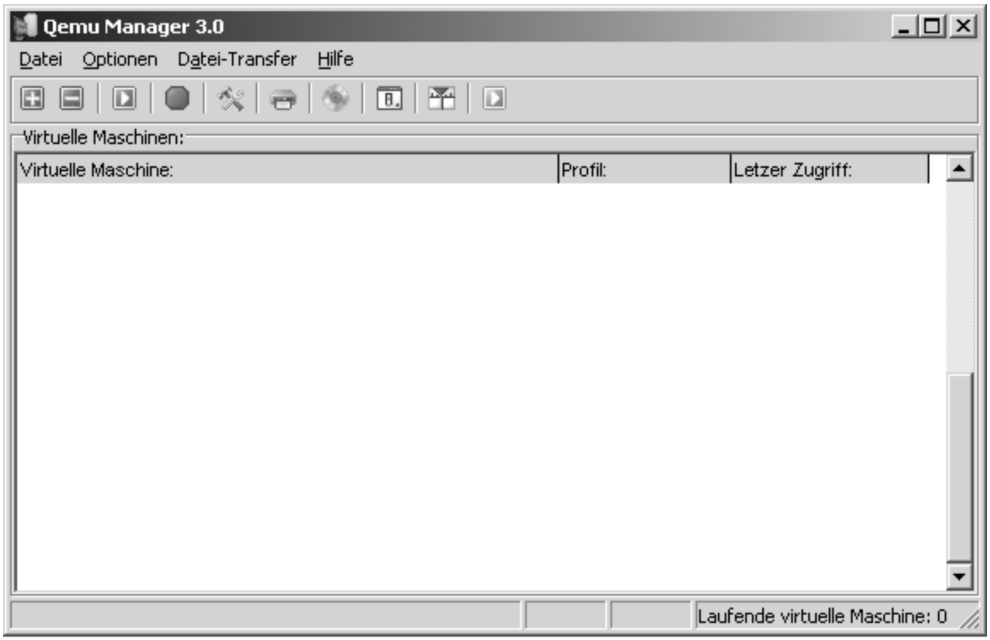

Abbildung 12: Der QEMU-Manager für Windows.

Nach dem Start des QEMU-Managers erscheint ein einfaches Fenster. Um eine neue virtuelle Maschine anzulegen, klickt man auf das rote Icon mit dem Plus-Zeichen ganz links in der Symbolleiste. Es startet ein Assistent zum Konfigurieren einer virtuellen Maschine, der die nötigen Informationen abfragt. Zuerst gibt man den Namen der neuen virtuellen

Maschinen an: *ReactOS*. Im nächsten Schritt wird die Betriebssystem-Vorgabe ausgewählt und der Arbeitsspeicher definiert. Für ReactOS sind die Vorgaben von Microsoft Windows XP auszuwählen und ein Arbeitsspeicher von 128 MByte einzustellen.

Danach wählt man den Punkt *Neues virtuelles Laufwerk erstellen* und legt damit eine neue virtuelle Festplatte an. Im Dateiauswahlfenster gibt man als Dateinamen *reactos* an. Der Name ist frei wählbar, sollte bei mehreren virtuellen Maschinen aber sinnvoll auf dem Namen des Betriebssystems in der Datei schließen lassen. Die Dateiendung *.dsk* wird automatisch angefügt. Im nachfolgenden Schritt wird die Größe der Festplatte definiert. Für den Test reicht 1 GByte. Als Typ ist *qcow2* zu empfehlen (siehe Abschnitt „Image-Formate“, Seite 67). *qcow2* ist das Dateiformat der aktuellen Version von QEMU.

Im Schritt *Basiseinstellungen der VM* wählt man das Einstellungs-Profil *Default* und lässt die Optionen *Audio verwenden* und *Full Screen verwenden* deaktiviert, da Audiounterstützung in ReactOS nicht unbedingt benötigt wird und die virtuelle Maschine in einem Fenster auf dem Desktop des Hostssystems ausgeführt werden soll.

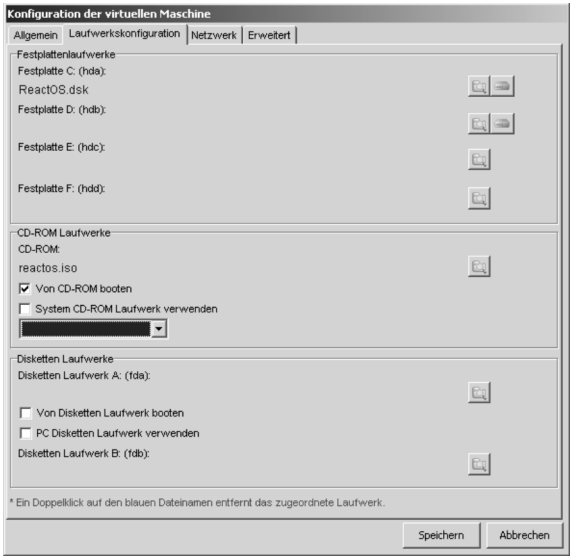

Abbildung 13: Das Einbinden der Installations-CD.

Um das Image der Installations-CD einbinden zu können, aktiviert man die Option *Erweiterte Konfigurationsoptionen nach dem Speichern anzeigen* und klickt anschließend auf der Button *Speichern der VM*. Es erscheint die Dialogbox zur erweiterten Konfiguration der virtuellen Maschine. Für die Installation von CD ist das CD-Image anzugeben. Dafür wählt man den Reiter *Laufwerkskonfiguration*. Unter *CD-ROM Laufwerke* wählt man die Image-Da-

tei der heruntergeladenen Installations-CD und aktiviert *Von CD-ROM booten*. Durch Klicken auf den Button *Speichern* werden die Einstellungen gesichert. Ist eine virtuelle Maschine erzeugt, erscheint ihr Name in der Liste. Durch Auswählen des Namens und durch Klicken auf das grüne Icon kann diese gestartet werden. Durch Klicken in das QEMU-Fenster werden die Maus- und Tastatur-Eingaben an das Gast-System übergeben. Das QEMU-Fenster wird mit [Strg]+[Alt] verlassen, zurück gelangt man wieder durch einen Klick in das QEMU-Fenster. Bei der Installation von ReactOS wird zuerst die deutsche Tastatur ausgewählt. Danach wird die virtuelle Festplatte partitioniert und formatiert. Die Vorgabe für das Zielverzeichnis und für den Bootloader werden übernommen. Danach muss der Computer neu gestartet werden. Nach erfolgreicher Installation ist das Booten von CD zu deaktivieren. Durch Doppelklick auf den Namen der virtuellen Maschine in der Liste gelangt man in das Konfigurationsfenster der virtuellen Maschine. Dort wechselt man zum Reiter *Laufwerkskonfiguration* und deaktiviert *Von CD-ROM booten*. Jetzt kann die virtuelle Maschine genutzt werden. Weiterführende Informationen zum Qemu-Manager für Windows findet man im Abschnitt „Grafische Benutzeroberflächen“ (Seite 131).

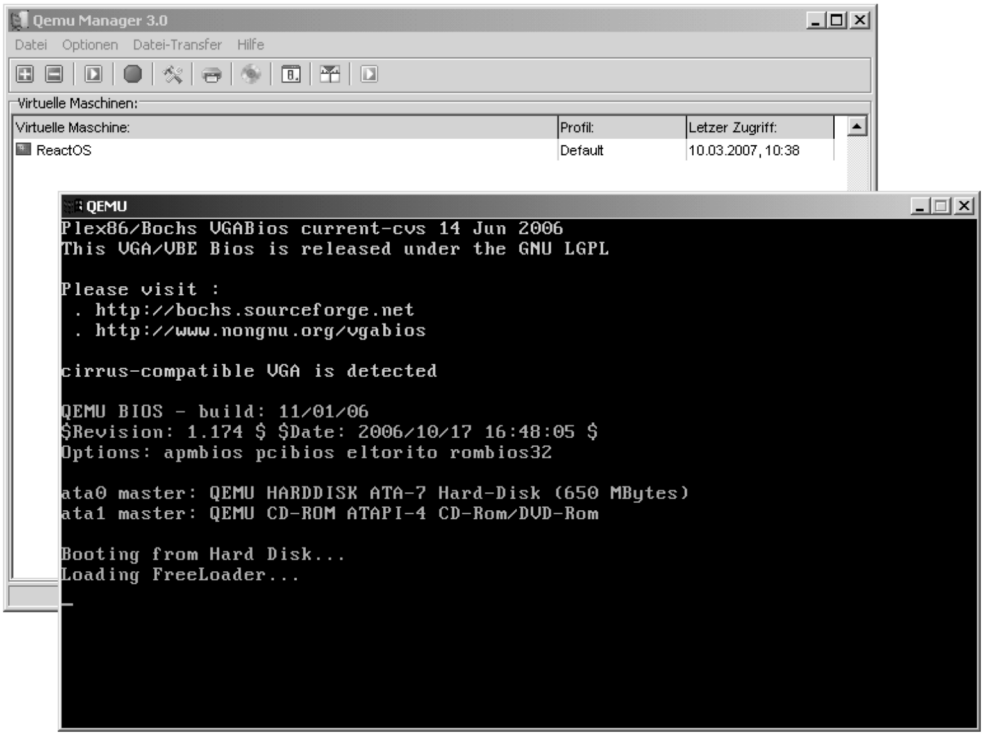

Abbildung 14: Der QEMU-Manager für Windows mit einer virtuellen Maschine.

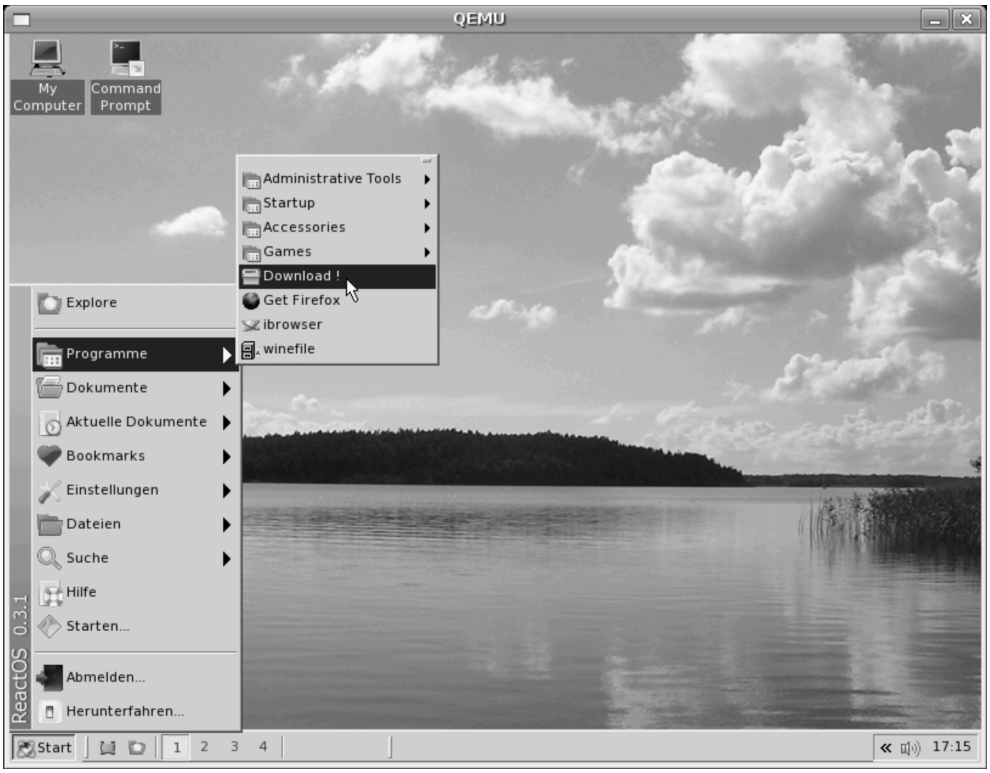

Abbildung 15: ReactOS unter QEMU.

Q (Mac OS X)

Unter Mac OS X wird QEMU vorzugsweise mit der grafischen Benutzeroberfläche *Q* gesteuert. Das Anlegen neuer virtueller Maschinen, das Modifizieren von Parametern sowie der Start der virtuellen Maschinen wird über *Q Control* durchgeführt.

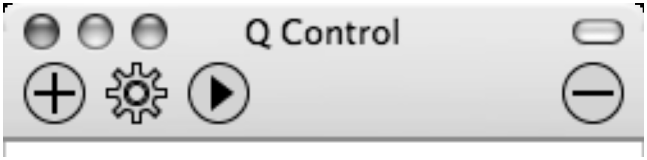

Abbildung 16: Die grafische Benutzeroberfläche Q Control.

Nach dem Start von *Q* wird automatisch das Kontrollzentrum für virtuelle Maschinen geöffnet. Damit lassen sich grundlegende Funktionen einrichten und nutzen. Um eine neue virtuelle Maschine anzulegen, klickt man das *+*-Symbol (*Neuen PC erstellen*) an. Danach erscheint ein Assistent, der alle relevanten Parameter abfragt. Im ersten Dialog wird der Name der virtuellen Maschine und der Typ des Gast-Systems festgelegt.

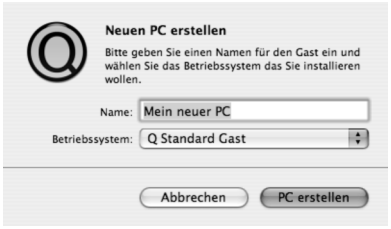

Abbildung 17: Q – Quickstart (Schritt 1).

In diesem Beispiel soll Microsoft Windows XP das Gast-Betriebssystem sein. Als Namen trägt man im Beispiel *WinXP* in das vorgesehene Eingabefeld ein. Er ist frei wählbar. Als Typ wird *Windows-XP* aus der Liste ausgewählt. Nach Betätigen von *PC erstellen* erscheint ein Dialogfenster zum Festlegen der Parameter und der emulierten Hardware.

Dieser Dialog weist die Karteikarten *Allgemein*, *Hardware*, *Netzwerk* und *Erweitert* auf. Auf der Karte *Allgemein* erscheint noch einmal der im vorigen Dialog vergebene Name der virtuellen Maschine. Unter *Emulation* werden der Zeitabgleich des Gast-Systems mit dem Host (*Zeit mit Uhr des Wirts abstimmen*) und das Netzwerk (*Netzwerk aktivieren*) aktiviert. Zum Dateiaustausch zwischen Host- und Gast-System dient der in *Q* integrierte Samba-

Server (SMB). SMB ist das in Windows-Netzwerken genutzte Protokoll. Unter *SMB Dateifreigabe Q Shared Files* wird der Austauschordner auf dem Host-System ausgewählt. Dieser Ordner wird von *Q* automatisch auf dem Desktop des angemeldeten Benutzers angelegt.

Abbildung 18: Q – Quickstart (Schritt 2).

Damit im Gast-System auf diesen Ordner zugegriffen werden kann, ist ein Treiber im Gast-System zu installieren, der die nötigen Einstellungen automatisch vornimmt und den Ordner als Laufwerk *Q* im Gastsystem einbindet. Die Installationsdatei wird in einem im Gastsystem eingeblendeten Laufwerk (der erste freie Laufwerksbuchstabe nach Festplatten und CDROM) aufgerufen. Dazu ist Diskette mit *Q Windows Treibern anzeigen* zu aktivieren. Nach der Installation dieses Treibers kann diese Einstellung wieder zurückgenommen werden. An dieser Stelle ist ein Blick in die Systemeinstellungen von Mac OS X nötig. Darin ist unter *Sharing* in der Karteikarte *Dienste* das *Windows Sharing* unbedingt auszuschalten, denn der in Mac OS X integrierte Samba-Server verträgt sich nicht mit dem in *Q* eingebauten Samba-Server. Ist der Samba-Server in Mac OS X aktiv, kann *Q* keine Verbindung zum Netzwerklaufwerk, also dem Dateiaustauschordner, herstellen.

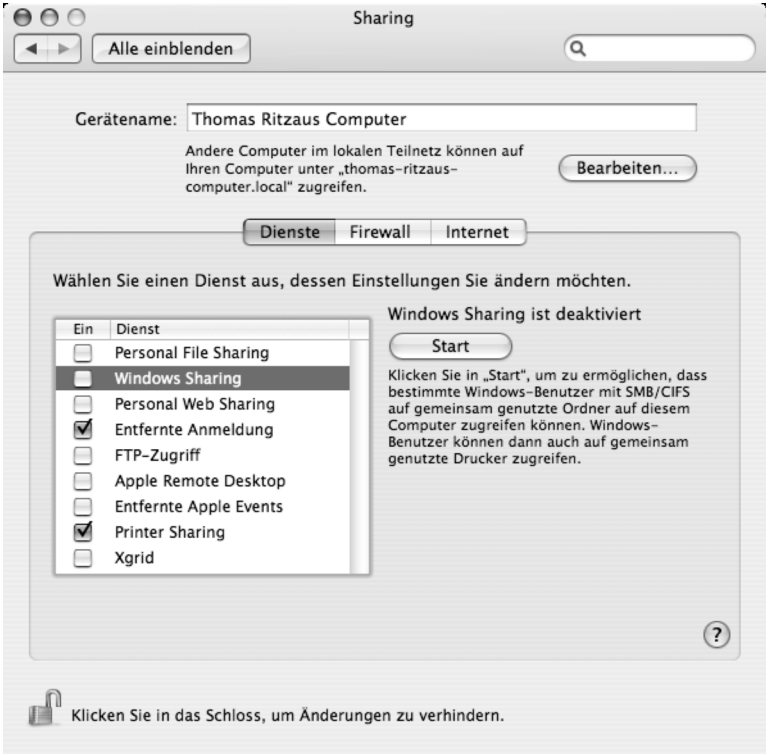

Abbildung 19: Windows Sharing ist in Mac OS X zu deaktivieren.

Die Karte *Hardware* dient zur Definition der für die virtuelle Maschine zu emulierenden Hardware-Komponenten.

Unter *Plattform* sollte bei der getesteten Version immer *x86PC* gewählt werden, denn der ebenfalls wählbare *x86-64 PC* ist instabil. Alle anderen Plattformen kommen für Microsoft Windows nicht in Frage. Die Anzahl der zu emulierenden CPUs kann ebenfalls eingestellt werden. Bei den modernen Rechnern mit zwei bis acht Kernel lassen sich hier auch Mehrprozessor-Systeme emulieren. Im Beispiel soll eine CPU genügen. Arbeitsspeicher sollte der virtuellen Maschine mit Microsoft Windows XP reichlich zugemessen werden, weniger als 512 MByte sind nicht empfehlenswert. Prinzipiell gelten hier dieselben Erfordernisse, wie sie auch an reale Maschinen gestellt werden.

Unter *Grafikkarte* wird eine Cirrus Logic GD5446 PCI VGA ausgewählt, denn der benötigte Treiber ist in Windows XP enthalten. Damit können Auflösungen jenseits Standard-VGA mit 16 Farben unter Microsoft Windows eingestellt werden. Die zur Auswahl stehenden Sound- und Netzwerkkarten sind Standardmodelle, die benötigten Treiber sind in Micro-

soft Windows XP enthalten. Die vorgeschlagene Konfiguration, *ENSONIQ AudioPCI ES 1370* als Soundkarte und *NE2000 PCI Netzwerk Adapter* als Netzwerkkarte, kann übernommen werden.

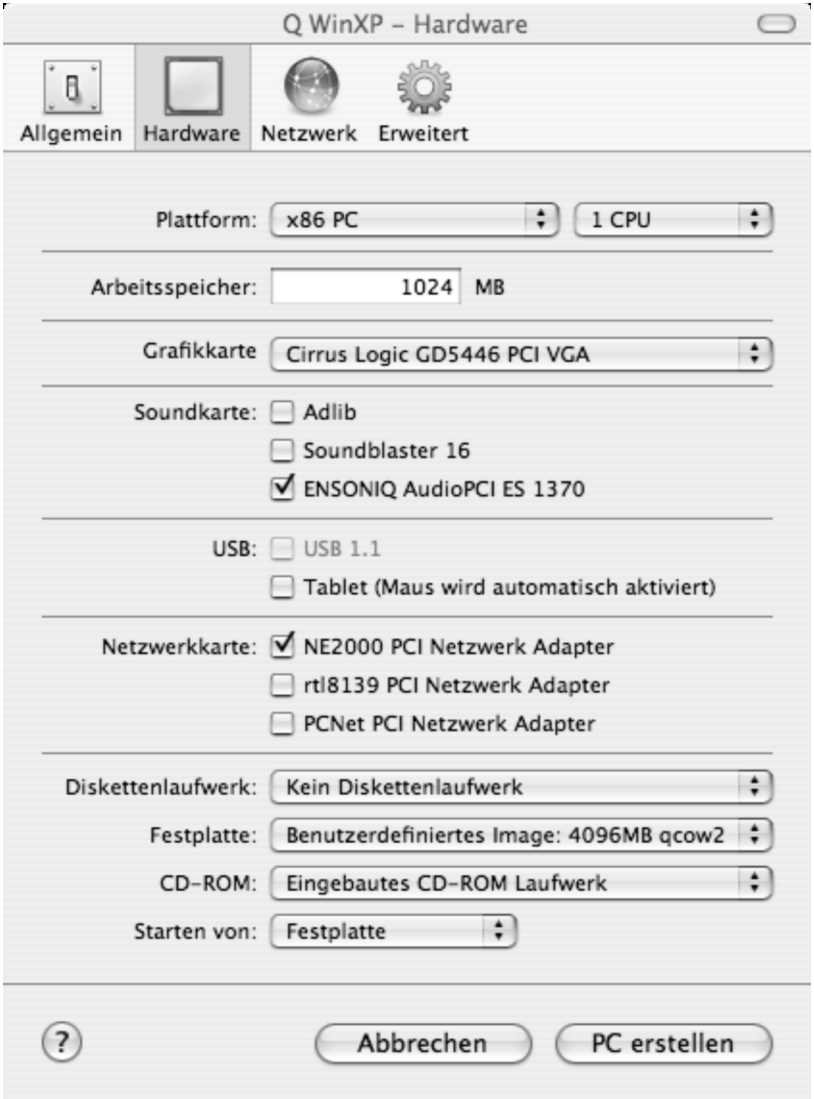

Abbildung 20: Q – Quickstart (Schritt 3).

Unter *USB* wird *Tablet* aktiviert, was ein virtuelles USB-Device als Maus-Ersatz bezeichnet. Mit diesem Gerät kann der Mauszeiger sowohl im Gast- als auch im Host-System genutzt werden. Verlässt der Mauszeiger das Fenster des Gast-Systems, wird automatisch auf den Zeiger des Host-Systems umgeschaltet (Maus wird automatisch aktiviert). Wird dagegen der Maustreiber von Microsoft Windows benutzt, ist der Mauszeiger im Fenster des Gast-Systems gefangen. Freigeben lässt er sich über die Tastenkombination [Strg]+[Alt].

Auf ein Diskettenlaufwerk ist verzichtbar (*Kein Diskettenlaufwerk*). Als Festplatte soll ein neues, im Beispiel vier GByte großes Image angelegt werden. Das ist zwar nicht groß, aber für ein erstes Testsystem sollte diese Größe reichen.

Die Installation des Gast-Systems von einer realen CD war auf Grund eines Bugs in der zum Zeitpunkt der Drucklegung vorliegenden Version von *Q* nicht möglich, weshalb hier ein ISO-Image der Windows-XP-Boot-CD eingehängt werden musste. Bootfähige Images vieler freier Betriebssysteme sind im Internet verfügbar. Von der eigenen Windows-CD muss man sich allerdings ein geeignetes Image erstellen, was allerdings einer gültigen Lizenz bedarf. Für jede mit Windows aufgesetzte virtuelle Maschine ist eine eigene Lizenz nötig! Für den ersten Start der virtuellen Maschine zur Installation des Gast-Systems ist *Starten von* auf *CD-ROM* einzustellen. Nach fertiggestellter Installation schaltet man hier wieder auf Festplatte um.

Abbildung 21: Q – Quickstart (Schritt 4).

Die Karteikarte *Netzwerk* kann für die erste virtuelle Maschine in der vorgegebenen Einstellung belassen werden. An dieser Stelle werden Einstellungen für die in QEMU enthaltene Firewall vorgenommen. Da in der ersten virtuellen Maschine keine von außen auf dem Gastsystem erreichbaren Dienste laufen sollen, bleiben erst einmal alle Optionen deaktiviert.

Auch die Karteikarte *Erweitert* bedarf keiner Änderung der vorgeschlagenen Optionen. Hier lassen sich weitere virtuelle Festplatten hinzufügen, weitere Parameter für den Start von QEMU vorgeben und diverse Besonderheiten von Linux und Windows 2000 festlegen. Dies alles wird für die erste virtuelle Maschine nicht benötigt.

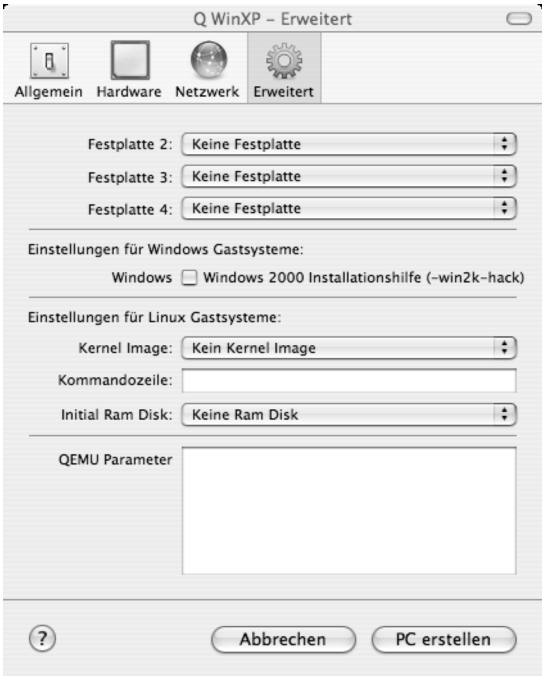

Abbildung 22: Q – Quickstart (Schritt 5).

Damit sind die Einstellungen für die erste virtuelle Maschine komplett. Mit *PC erstellen* wird die virtuelle Maschine vervollständigt. Sie erscheint in der Liste der verfügbaren virtuellen Maschinen und kann durch Doppelklick auf den Namen gestartet werden. Die neue virtuelle Maschine bootet vom Image der Installations-CD und führt die gewohnte Betriebssystem-Installation durch. Dabei gibt es keine Besonderheiten im Vergleich mit einem realen PC. Der virtuelle PC verhält sich bei der Installation dank seiner Standardkomponenten eher unauffällig.

Abbildung 23: Q – Die virtuelle Maschine ist fertiggestellt.