JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Virtuelle Maschinen

SEMINARARBEIT
Seminar Betriebssysteme
LVA-Nr. 353.062
SS2005

Eingereicht von:
Markus Pfleger, 0255760

Angefertigt am:
Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM)

Betreuung:
Prof. Mühlbacher
DI Andreas Putzinger

Linz, Februar 2005

Abstract

Incompatibility between computer systems is getting a real problem, caused by the increasing level of networking. Nearly every modern computer system has access to the internet. This not only gives access to a huge amount of data, but also to a lot of software. But why can't every software be used from every computer system like it is the case with data?

The reason is the incompatibility of the interfaces of different systems. One solution to this problem is the concept of virtual machines. This paper discusses the different types of virtual machines and how they work. Additionally there are some examples of virtual machines that are compared to each other.

Kurzfassung

Inkompatibilität zwischen unterschiedlichen Computersystemen wird, mit zunehmender Vernetzung, immer mehr zu einem Problem. Praktisch jedes moderne Computersystem hat Zugang zum Internet und damit nicht nur Zugang zu einer riesigen Menge an Daten, sondern auch zu einer riesigen Menge an Software. Dabei stellt sich die Frage, warum Software nicht, ebenso wie Daten, auf jedem Computersystem eingesetzt werden kann.

Der Grund dafür ist die Inkompatibilität der Schnittstellen von verschiedenen Systemen. Eine mögliche Lösung ist das Konzept der virtuellen Maschinen. In dieser Arbeit soll darauf eingegangen werden, inwiefern virtuelle Maschinen diese Inkompatibilitäten aufheben können, welche unterschiedlichen Arten von virtuellen Maschinen es gibt und wie diese arbeiten. Außerdem werden verschiedene Beispiele von virtuellen Maschinen miteinander verglichen.

Inhaltsangabe

1. Einleitung

Ein großes Problem im Bereich der EDV ist, dass unterschiedliche Computersysteme häufig nicht, bzw. nur unzureichend kompatibel zueinander sind. Das bedeutet, dass Software, welche für ein bestimmtes System geschrieben wurde, nicht einfach auf einem anderen System ausgeführt werden kann. Das Problem dabei sind inkompatible Schnittstellen. Schnittstellen sind Berührungspunkte zwischen verschiedenen Komponenten eines Systems. Diese müssen zusammenpassen, um miteinander arbeiten zu können.

Es gibt dabei unterschiedliche Arten von Schnittstellen:

Hardwareschnittstellen

Eine Hardwareschnittstelle ist eine Schnittstelle zwischen zwei physischen Systemen. Dabei besteht eine Hardwareschnittstelle meist aus zwei Teilen. Der physischen Schnittstelle (Stecker, Pin Belegung bei Anschlüssen,…) und der logischen Schnittstelle (dem Übertragungsprotokoll). Diese Schnittstellen sorgen dafür, dass unterschiedliche physische Systeme zusammenarbeiten können.
Beispiel für Hardwareschnittstellen sind in Computersystemen beispielsweise PCI-Bus, SCSI, SATA,… . An eine SATA Schnittstelle kann dabei jede beliebige Festplatte von jedem Hersteller angeschlossen werden, solange auch diese eine SATA Schnittstelle aufweist. Dabei muss natürlich sowohl die physische Schnittstelle als auch die logische Schnittstelle übereinstimmen.
[Wiki]

Um physische Systeme miteinander zu verbinden, die nicht über eine gemeinsame Schnittstelle verfügen, können Adapter verwendet werden. Adapter besitzen dabei im Allgemeinen 2 Schnittstellen unterschiedlichen Typs und sorgen damit dafür, dass 2 eigentlich inkompatible Geräte zusammenarbeiten können. So könnte man zum Beispiel einen Adapter benutzen, um eine IDE Festplatte an eine SATA Schnittstelle anzuschließen.

Softwareschnittstellen

Softwareschnittstellen sind logische Berührungspunkte in einem Softwaresystem. Diese definieren, wie Kommandos und Daten zwischen verschiedenen Prozessen und Komponenten ausgetauscht werden. Softwareschnittstellen kommen in Computersystemen an verschiedenen

Stellen vor. Diese sind beispielsweise Systemschnittstellen zum Zugriff auf Systemroutinen, Schnittstellen zur Kommunikation mit anderen Prozessen und Schnittstellen zum Verbinden einzelner Softwarekomponenten (Module) eines Programms.
[Wiki]

Um Software miteinander zu verbinden, die über keine gemeinsame Schnittstelle verfügt kann man auch wieder einen Adapter verwenden. Im Zusammenhang mit Software werden im Allgemeinen allerdings andere Ausdrücke verwendet. So bezeichnet man einen Adapter, um Softwarekomponenten eines Programms zueinander kompatibel zu machen meist als Wrapper. Adapter für Systemschnittstellen werden hingegen meist als VMM (Virtual Machine Manager) bezeichnet, das darunter liegende System inklusive VMM als virtuelle Maschine.

1.1. Aufbau eines Computersystems

Um die Hauptteile eines Computersystems unabhängig voneinander entwickeln zu können bzw. Teile problemlos austauschen zu können wurde bereits in den frühen 60er Jahren damit begonnen, eine Schnittstelle zwischen Hard- und Software exakt zu definieren. Diese Schnittstelle wird als ISA (Instruction Set Architecture) bezeichnet.

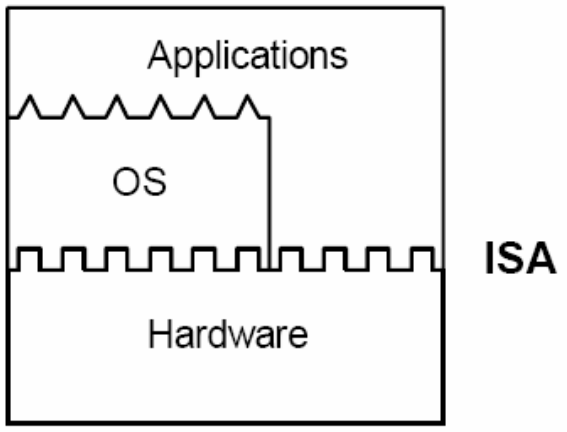

Abbildung 1: Aufbau eines Computersystems [VMA]

Die ISA ist eine Schnittstelle zwischen Hard- und Software. Sämtliche Software, also sowohl das Betriebssystem als auch die Anwendungen, greift auf die Hardware über diese definierte Schnittstelle zu. Die Vorteile eines solchen Konzeptes liegen dabei auf der Hand. Einerseits

können Hardware und Softwarehersteller weitgehend unabhängig voneinander entwickeln und andererseits ist es jederzeit möglich einen Teil auszutauschen, ohne dass die anderen Teile davon in Mitleidenschaft gezogen werden. Es kann also eine völlig neue Hardware Plattform verwendet werden und es ist trotzdem möglich, bestehende Software weiter zu verwenden. Vorraussetzung dafür ist natürlich, dass dabei die ISA Schnittstelle unverändert bleibt.

Dieses Konzept bringt allerdings auch einige Probleme mit sich. Ein Problem ist, dass nicht jede Hardware Plattform die gleiche ISA Schnittstelle benutzt. So hat beispielsweise ein PowerPC eine andere ISA Schnittstelle als eine x86 Architektur. Ein weiteres Problem besteht zwischen Anwendungssoftware und dem Betriebssystem, also der Systemschnittstelle. Greift eine Anwendung auf Betriebssystemfunktionen zu, so ist es nicht mehr möglich, einfach das Betriebssystem auszutauschen, da unterschiedliche Betriebssysteme zueinander inkompatible Schnittstellen anbieten.

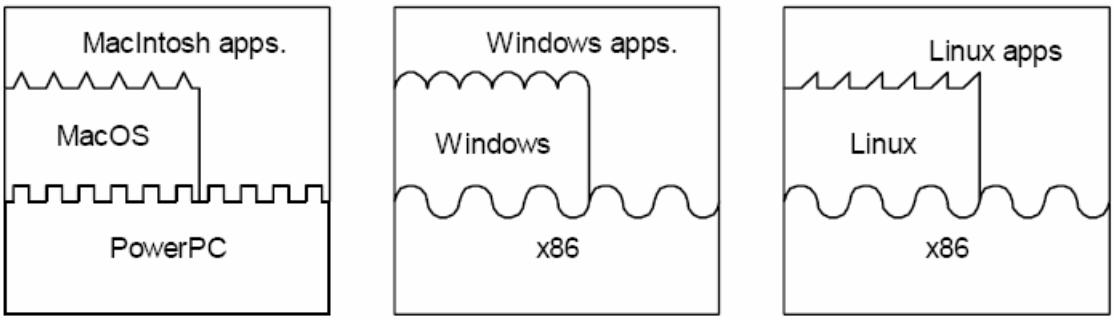

Abbildung 2: Unterschiedliche Architekturen mit verschiedenen Schnittstellen [VMA]

Wie man in Abbildung 2 sehen kann, haben nicht nur die Hardwarearchitekturen PowerPC und x86 eine unterschiedliche Schnittstelle, sondern auch die Betriebssysteme MacOS, Windows und Linux.
Es ist also beispielsweise nicht möglich das Betriebssystem MacOS auf einer x86 Architektur zu installieren. Außerdem ist es auch nicht möglich, eine Windows Applikation auf einem Linux System zu installieren.

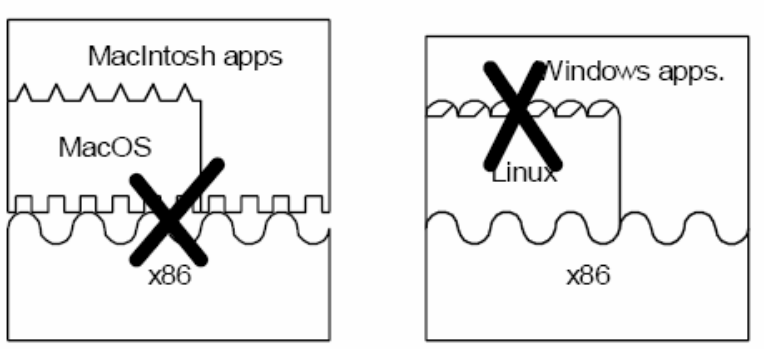

Abbildung 3: Systeme mit nicht kompatiblen Schnittstellen [VMA]

Ein weiteres Problem ist es, dass Verbesserungen an einer Komponente (also Hardware oder Software) von einer veralteten ISA Schnittstelle behindert werden können.

Diese Probleme können durch eine Softwareschicht, welche zwischen den entsprechenden Schnittstellen, als eine Art Adapter verwendet wird, gelöst werden. Ein System, welches eine solche Softwareschicht besitzt, wird als **„virtuelle Maschine“** bezeichnet.
[VMA]

2. Virtuelle Maschinen

2.1. Was ist eine virtuelle Maschine

Eine virtuelle Maschine ist ein System, in dem zwischen 2 inkompatiblen Schnittstellen eine Softwareschicht, die so genannte „virtualisierende Software“, als eine Art Adapter, eingesetzt wird. Die Schichten unter dieser Softwareschicht werden dabei als Gastgeber (Host) und die darüber liegenden Schichten als Gast (Guest) bezeichnet. Der Begriff virtuelle Maschine bezeichnet dabei den Gastgeber (Host) und die „virtualisierende Software“.

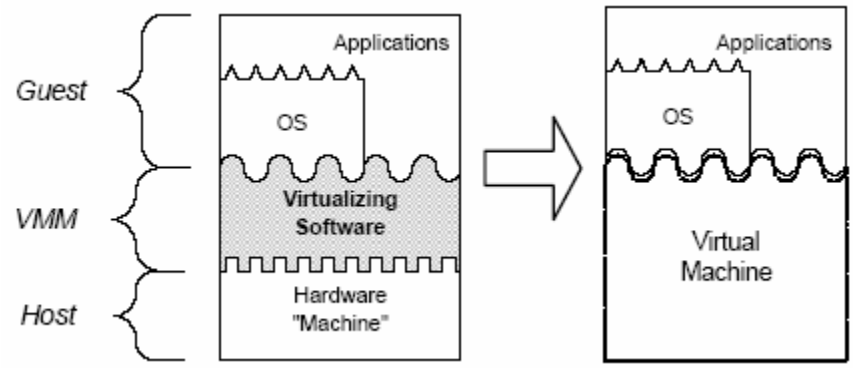

Abbildung 4: Eine Softwareschicht vermittelt zwischen Betriebssystem und Anwendungssoftware auf der einen und der Hardware auf der anderen Seite [VMA]

In Abbildung 4 sehen Sie eine Art von virtueller Maschine. Bei dieser Art von virtueller Maschine wird die „virtualisierende Software“ zwischen unterschiedlichen ISA Schnittstellen eingezogen. Dies hat zur Folge, dass die darüber liegenden Schichten, also das Betriebssystem und die Anwendungssoftware, eine andere ISA sehen als jene, die von der Hardware unterstützt wird. Diese Schichten sehen nur die virtuelle Maschine, nicht jedoch die tatsächliche Hardware. Es ist dann die Aufgabe der „virtualisierenden Software“, Hardwareaufrufe von den oberen Schichten so umzuwandeln, dass sie der ISA Schnittstelle der tatsächlichen Hardware entsprechen.
Die hier gezeigte Virtualisierung der ISA Schnittstelle ist dabei nur eine Möglichkeit der Virtualisierung. Andere Möglichkeiten lernen Sie im Abschnitt „Arten von virtuellen Maschinen“ kennen. [VMA]

2.2. Arten von virtuellen Maschinen

Es gibt viele verschiedene Arten von virtuellen Maschinen. Man kann diese aber grob in 2 Hauptkategorien einteilen. Diese sind **Prozess virtuelle Maschinen** und **System virtuelle Maschinen.** Der Unterschied liegt dabei darin, welche Schnittstelle virtualisiert wird. Es kann dies einerseits die ABI (Application Binary Interface) Schnittstelle und andererseits die ISA (Instruction Set Architecture) Schnittstelle sein.

Bei einer Prozess virtuellen Maschine wird dabei die ABI Schnittstelle (oder Teile davon) virtualisiert, was dazu führt, dass der Anwendungssoftware (bzw. einem Prozess) eine andere Umgebung vorgetäuscht wird, als die tatsächlich vorhandene. Der Prozess arbeitet also auf einer virtuellen Maschine.

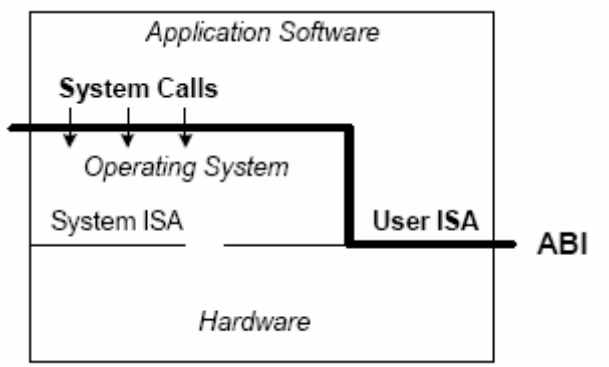

Abbildung 5: Die ABI (Application Binary Interface) Schnittstelle [VMA]

Auf diese Weise ist es möglich eine Anwendung Plattformunabhängig zu machen. Je nach Art der Prozess virtuellen Maschine ist es so möglich, beispielsweise ein Programm sowohl unter Windows auf einer x86 Architektur, als auch unter Linux auf einer x86 Architektur, aber auch unter MacOS auf einer PowerPC Architektur ablaufen zu lassen. Der Prozess hat dabei keine Kenntnis von der jeweiligen Architektur.
[VMA]

Bei einer System virtuellen Maschine wird im Gegensatz dazu die ISA Schnittstelle virtualisiert, was dazu führt, dass nicht nur ein Prozess, sondern das ganze System glaubt auf einer Hardware zu arbeiten, die physikalisch nicht existiert. In diesem Fall arbeitet also ein gesamtes System auf einer virtuellen Maschine.

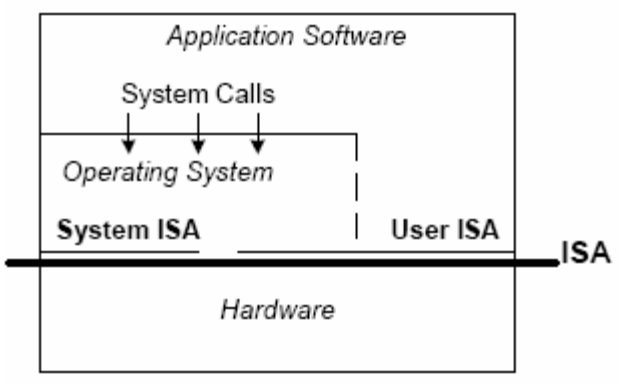

Abbildung 6: Die ISA (Instruction Set Architecture) Schnittstelle [VMA]

Durch die Virtualisierung der ISA Schnittstelle kann einem ganzen System, also Betriebssystem und Anwendungssoftware, eine andere Umgebung vorgetäuscht werden. Es ist auf diese Weise beispielsweise möglich, Windows mit allen möglichen Windows Anwendungen auf einer PowerPC Architektur zu installieren. Das System hat dabei keine Kenntnis von der zugrunde liegenden Hardware.
[VMA]

2.3. Prozess virtuelle Maschinen

Eine Prozess virtuelle Maschine stellt der Anwendungssoftware eine virtuelle ABI Schnittstelle zur Verfügung. Es gibt dabei zahlreiche Implementierungen von Prozess virtuellen Maschinen. Die wichtigsten dabei sind in den nächsten Abschnitten näher erläutert.

2.3.1. Multiprogrammierung

Multiprogrammierung ist die wohl am weitest verbreitete virtuelle Maschine ist ein Teil eines jeden modernen Betriebssystems und wird meistens auch nicht als virtuelle Maschine bezeichnet. Es handelt sich bei der Multiprogrammierung deshalb um eine virtuelle Maschine, weil vom Betriebssystem jedem Prozess der Eindruck vermittelt wird, dass er alleine den Rechner zur Verfügung hat.
[VMA]

2.3.2. Emulation und Binärumwandlung

Bei dieser Art von Prozess virtueller Maschine, wird ein kompiliertes Programm, welches auf einer speziellen ISA Schnittstelle kompiliert wurde, auf einer anderen ISA Schnittstelle zur Ausführung gebracht. Es handelt sich aber trotzdem um eine Prozess virtuelle Maschine, weil lediglich der Prozess mit einer eigentlich nicht vorhandenen Maschine arbeitet. Das Betriebssystem muss hingegen für die entsprechende ISA Schnittstelle gemacht sein. Es wird also nur für die entsprechende Anwendungssoftware ein Adapter, zwischen der realen ISA Schnittstelle und der Schnittstelle die von der Anwendungssoftware benutzt wird, eingesetzt.

Ein Beispiel für eine solche Virtuelle Maschine ist das Digital FX!32 System. Dieses System kann Programme, welche für eine x86 Architektur (Source-ISA) und Windows NT kompiliert wurden, auf einer Alpha Architektur (Target-ISA) mit Windows NT ausführen.

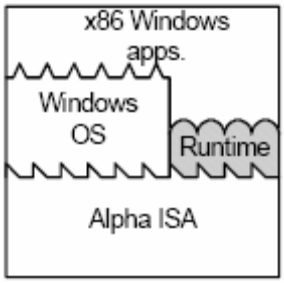

Abbildung 7: Eine so genannte Runtime (die virtualisierende Software) arbeitet als Adapter zwischen den x86 Windows Applikationen und der Alpha ISA Schnittstelle [VMA]

Wie man in Abbildung 7 erkennen kann, ist zwischen der Hardware und einer x86 Windows Applikation die „virtualisierende Software“ in Form der Runtime vorhanden. Zwischen dem Betriebssystem und der x86 Windows Applikation wird dabei kein Adapter benötigt, weil Windows NT für x86 die gleiche Schnittstelle wie Windows NT für Alpha realisiert.

Die Aufgabe dieser Runtime ist es, der x86 Windows Applikation nach oben hin eine ISA Schnittstelle zur Verfügung zu stellen, welche der Source-ISA entspricht, für die das Programm kompiliert wurde. Nach unten hin, muss diese Runtime allerdings die ISA Schnittstelle der Alpha Architektur verwenden. Es müssen also letztendlich Anweisungen der x86 Windows Applikation, in Anweisung umgewandelt werden, welche von der Alpha ISA Schnittstelle verstanden werden.

Um diese Aufgabe zu erfüllen, gibt es 2 grundlegend verschiedene Verfahren. Einerseits die Interpretation und andererseits die Binärumwandlung.

Bei der Interpretation wird die Applikation von einem Interpreter ausgeführt. Dieser Interpreter dekodiert dabei die Instruktionen welche von der Applikation aufgerufen werden und emuliert die Ausführung dieser Instruktionen der Source-ISA mittels Instruktionen, welche von der Target-ISA angeboten werden. Das kann allerdings sehr langsam sein, weil jede Instruktion der Applikation erst interpretiert werden muss bevor sie ausgeführt werden kann.

Die zweite Möglichkeit, welche im Allgemeinen die bessere Performance liefert, ist die Binärumwandlung. Bei der Binärumwandlung werden ganze Blöcke von Instruktionen der Applikation in native Instruktionen der Target-ISA umgewandelt. Diese Umwandlung kann

einige Zeit in Anspruch nehmen. Wenn dann allerdings die nativen Instruktionen zur Verfügung stehen, dann können diese gecached und in weiterer Folge sehr schnell ausgeführt werden. Es muss dann nicht mehr jede einzelne Instruktion interpretiert werden.
[VMA]

2.3.3. Virtuelle Maschinen für Hochsprachen

Virtuelle Maschinen, welche für moderne Hochsprachen, wie beispielsweise Java oder C#, verwendet werden gehören ebenfalls in die Kategorie der Prozess virtuellen Maschinen. Diese gehen dabei aber in eine etwas andere Richtung als die bisher diskutierten Ansätze. Bisher wurde ein Programm für ein bestimmtes System mit einer bestimmten Schnittstelle kompiliert (z.B. für ein x86 Windows System) und dann durch eine virtuelle Maschine auch auf anderen Systemen zum laufen gebracht.

Das ist aber eine relativ schwierige Angelegenheit, vor allem dann, wenn ein Programm auf den verschiedensten Plattformen, mit unterschiedlichen Betriebssystemen laufen soll.

Wesentlich einfacher ist es dabei, nicht nur eine virtuelle Maschine zu entwickeln, sondern gleichzeitig eine Programmierumgebung, welche die damit erstellten Programme nicht für ein bestimmtes System in Maschinencode kompiliert, sondern erst in eine Zwischensprache. Diese Zwischensprache ist dabei so ausgelegt, dass es möglichst einfach ist, ein Programm in dieser Zwischensprache auf den verschiedensten Systemen, mittels einer virtuellen Maschine, auszuführen.

Der Vorteil dieses Konzeptes ist es, dass diese Zwischensprache dem Maschinencode sehr ähnlich ist, allerdings wesentlich abstrakter. Es ist deshalb sehr viel einfacher, eine virtuelle Maschine zu erstellen, welche diesen Zwischencode auf einem bestimmten System ausführt, als einen Compiler zu erstellen, welcher direkt die Hochsprache in Maschinencode für das jeweilige System umwandelt. Andererseits ist es auch wesentlich einfacher, eine virtuelle Maschine für diesen abstrakten Zwischencode zu erstellen, als eine virtuelle Maschine für eine ISA Schnittstelle, wie sie tatsächlich verwendet wird zu machen.
Bei virtuellen Maschinen für Hochsprachen wird also die „virtualisierende Software“ bzw. Runtime zwischen den Anwendungsprogrammen und den darunter liegenden Schichten (also Betriebssystem und Hardware) eingezogen. Es entsteht auf diese Art also ein Adapter, der

über die gesamte ABI Schnittstelle geht. Die Runtime übersetzt also alle Aufrufe der virtuellen ABI Schnittstelle, in Aufrufe für die konkret vorhandene ABI Schnittstelle umwandeln.

Für diese Umwandlung stehen auch, wie bereits im letzten Punkt beschrieben, die Methoden der Interpretation bzw. der Binärumwandlung, sowie Abwandlungen bzw. Kombinationen davon zur Verfügung.
[VMA]

2.4. System virtuelle Maschinen

Eine System virtuelle Maschine stellt der Software eine virtuelle ISA Schnittstelle zur Verfügung. Dabei wird von einer System virtuellen Maschine eine ganze Systemumgebung vorgetäuscht, die in der Form eigentlich gar nicht existiert.
Es gibt dabei zahlreiche Implementierungen von System virtuellen Maschinen. Die wichtigsten dabei sind in den nächsten Abschnitten näher erläutert.

2.4.1. Klassische System virtuelle Maschinen

Durch eine klassische System virtuelle Maschine ist es möglich, auf einer Hardware, mehrere Betriebssysteme zu betreiben. Die Hauptaufgabe der virtuellen Maschine ist dabei, die Hardwareressourcen zwischen den verschiedensten Gast Betriebssystemen aufzuteilen. Dabei wird zwischen der Hardware und der Software (Betriebssystem und Anwendungssoftware – also auf Ebene der ISA) die „virtualisierende Software“ eingesetzt, welche in diesem konkreten Fall unter dem Namen VMM (Virtual Maschine Monitor) bekannt ist.

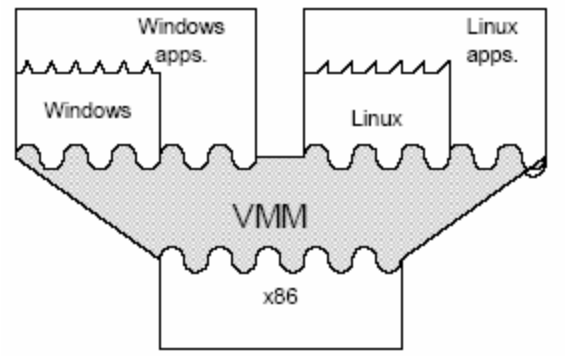

Abbildung 8: Eine System virtuelle Maschine, wobei die VMM direkt auf der Hardware sitzt [VMA]

Diese Art der System virtuellen Maschine ist im Allgemeinen die effizienteste und bietet dabei auch allen Gastsystemen die gleichen Möglichkeiten. Das Hauptproblem dieses Ansatzes ist vor allem, dass die VMM direkt auf der Hardware sitzt und deshalb zuerst die VMM und dann die Gastsysteme installiert werden müssen. Außerdem müssen auch die Gerätetreiber in der VMM installiert werden, weil es die VMM ist die direkt mit den I/O Geräten kommuniziert.

Dieser Ansatz entspricht im Wesentlichen dem der Multiprogrammierung, allerdings auf Systemebene. Es wird also nicht den einzelnen Prozessen vorgespielt dass sie die Hardware alleine benutzen, sondern es wird den ganzen System vorgespielt dass sie die Maschine für sich alleine haben.

Wie man in Abbildung 8 sehen kann, wird für diesen Typ von virtueller Maschine auf der gleichen Architektur, also auf der gleichen ISA Schnittstelle gearbeitet. Das heißt, dass auf einer solchen System virtuellen Maschine nur Gastbetriebssysteme installiert werden können, die ebenfalls diese ISA Schnittstelle verwenden. Das hat aber andererseits den Vorteil, dass die VMM keine Interpretation oder Übersetzung der Instruktionen vornehmen muss. Dies wirkt sich natürlich positiv auf die Performance eines solchen Systems aus.
[VMA]

Gehostete virtuelle Maschinen

Das Prinzip dieser Art von System virtuellen Maschinen ist das gleiche, wie bei den klassischen System virtuellen Maschinen. Der Unterschied dabei liegt daran, dass bei einer geshosteten virtuellen Maschine die VMM Software innerhalb eines existierenden Host Betriebssystems installiert wird und nicht direkt auf der Hardware aufsetzt. Das hat den Vorteil, dass die Installation der VMM Software, der Installation eines Anwendungsprogrammes gelicht. Außerdem kann die VMM Software so auf die Gerätetreiber des Host Betriebssystems zurückgreifen.

Der Nachteil dabei ist allerdings, dass diese Art der Implementierung nicht so effizient ist, wie eine klassische System virtuelle Maschine.
[VMA]

2.4.2. Emulator (Whole System Virtual Machines)

Bei dieser Art von virtueller Maschine ist es möglich, dass die unterschiedlichen Betriebssysteme und Anwendungsprogramme eine unterschiedliche ISA Schnittstelle verwenden.

Dabei muss die VMM als Adapter zwischen den unterschiedlichen Schnittstellen tätig werden und die Instruktionen entsprechend interpretieren bzw. Übersetzen. Das kann sehr aufwendig sein, weil die gesamte Hardwareumgebung, welche vom Gastsystem vorausgesetzt wird, emuliert werden muss. Für ein solches System ist eine Interpretation aus Performancegründen nicht mehr praktikabel. Aber auch mittels Binärübersetzung kann es zu sehr großen Performanceverlusten kommen.

Der Vorteil ist natürlich, dass auf diese Weise Systeme auf einer Hardware laufen können, für die sie gar nicht gedacht waren. Ein Beispiel dafür ist, dass so auf einem PowerPC auch ein Windows System installiert werden kann. Dabei wird im Allgemeinen der Ansatz umgesetzt, dass die VMM auf einem bereits existierenden Betriebssystem aufsetzt.

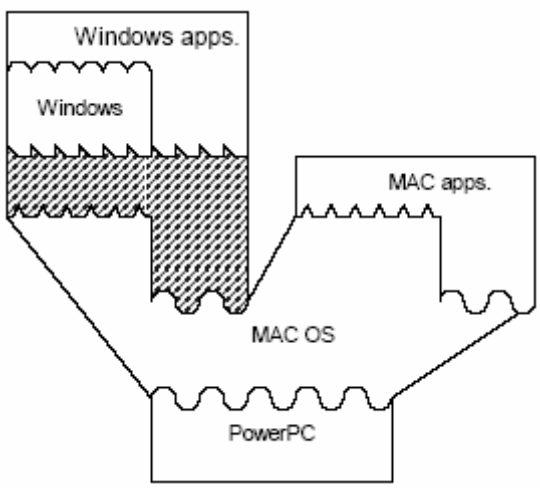

Abbildung 9: Ein Windows System, dass innerhalb eines MacOS auf einm PowerPC läuft [VMA]

[VMA]

3. Beispiele für virtuelle Maschinen

3.1. Überblick

	Java VM	.NET CLR	VMWare	PearPC
Kategorie	PVM	PVM	SVM	SVM
Klasse	HLL	HLL	(gehostete) klassische VM	Emulator
Gleiche ISA	nein	nein	ja	nein
Ausführung	interpretiert/nativ	nativ	nativ	emuliert
Geschwindigkeit	mittel/gut	gut	(gut) sehr gut	schlecht

Legende

PVM: Prozess Virtuelle Maschine

SVM: System Virtuelle Maschine

HLL: High Level Language (virtuelle Maschine für Hochsprachen)

3.2. Java

Bei Java handelt es sich um eine Programmiersprache, genauer gesagt eine Hochsprache (High Level Language), die sehr eng mit der „Java Virtual Machine“ zusammenhängt. Wie der Name schon sagt, handelt es sich bei der „Java Virtual Machine“ um die virtuelle Maschine, für welche der Java Programmcode kompiliert wird. In diesem Paper werden nur die grundlegenden Aspekte über die Java VM angesprochen. Detaillierte Informationen zu diesem Thema finden Sie bei [JAN].

3.2.1. Klassifikation

Entsprechend den Arten von virtuellen Maschinen, die in Punkt 2.2 diskutiert wurden, handelt es sich bei der „Java Virtual Machine“, oder kurz JVM, um eine so genannte Prozess virutelle Maschine. Genauer gesagt um eine virtuelle Maschine für Hochsprachen, wie sie in Punkt 2.3.3 besprochen wird.

Für ein Java Programm, dass in der JVM ausgeführt wird, ist es so völlig transparent, ob es auf einer x86 oder einer PowerPC Architektur ausgeführt wird. Außerdem ist es ebenfalls transparent, ob sich auf der jeweiligen Architektur ein Windows, Linux oder MacOS Betriebssystem befindet.

3.2.2. Wie entsteht ein Java Programm

Programme die in Java geschrieben sind, werden, wie bei den meisten anderen Programmiersprachen auch, zunächst mit einem Compiler kompiliert. Der Unterschied zu anderen Programmiersprachen wie C oder C++ liegt darin, dass durch diese Kompilation kein Maschinencode erzeugt wird, der an eine konkret verfügbare Hardware (also eine spezielle ABI-Schnittstelle) gebunden ist. Der Code der durch den Compiler erzeugt wird, ist vielmehr der Maschinencode für die JVM, also ein abstrakter Maschinencode. Dieser Code wird bei Java Bytecode gnannt.

Für die Ausführung eines so kompilierten Java Programms, wird dann eben jene Maschine, die „Java Virtual Maschine“ benötigt. Der Vorteil von dieser Vorgehensweise ist, dass damit jedes Java Programm auf jedem beliebigen System laufen kann, vorausgesetzt es existiert eine Implementierung der JVM für dieses System.
[CVM]

3.2.3. Arbeitsweise der JVM

Die „Java Virtual Maschine“ arbeitet, wie bei solchen virtuellen Maschinen üblich, als Adapter zwischen der Anwendungssoftware und den darunter liegenden Schichten (Betriebssystem und Hardware).

Für das Ausführen des Bytecodes setzen die meisten JVM’s, aus Performancegründen, einen sogenannten JIT-Compiler ein (Just in time Compiler). Bei einem solchen JIT-Compiler wird der Bytecode, während des Programmablaufs, in den entsprechenden Maschinencode übersetzt. Auf eine reine Interpretation des Bytecodes wird meist verzichtet, weil diese sehr langsam ist.
[CVM]

3.3. .NET

Bei .NET handelt es sich um eine Technologie, welche der von Java in weiten Teilen sehr ähnelt. Es ist ebenfalls ein Paket aus Programmiersprachen und einer virtueller Maschine, der so genannten CLR (Common Language Runtime). Dabei steht eine ganze Menge an Programmiersprachen zur Verfügung, die mit dieser virtuellen Maschine zusammenarbeiten. Diese sind unter anderem C++.NET, C#, ASP.NET, … In diesem Paper werden nur die grundlegenden Aspekte über die .NET CRL angesprochen. Detaillierte Informationen zu diesem Thema finden Sie bei [JAN].

3.3.1. Klassifikation

Bei der CLR von .NET handelt es sich, wie bei der „Java Virtual Machine“, um eine Prozess virtuelle Maschine (siehe Punkt 2.2) bzw. genauer um eine virtuelle Maschine für Hochsprachen, wie sie unter Punkt 2.3.3 besprochen wird.

Ein .NET Programm ist also, vom Prinzip her, ebenso plattformunabhängig wie ein Java Programm. Allerdings stehen wesentlich weniger Implementierungen der CLR zur Verfügung als Implementierungen der JVM.

3.3.2. Wie entsteht ein .NET Programm

Ähnlich Java, wird auch ein .NET Programm mithilfe eines Compilers zunächst in eine Zwischensprache, die CIL (Common Intermediate Language), kompiliert. Diese Sprache kann man sich als eine Art Maschinencode für die CLR vorstellen. Diese Zwischensprache ist dabei ebenfalls darauf ausgelegt, unabhängig von der konkreten Plattform zu sein, auf der das Programm später ausgeführt wird.

Um ein Programm, welches in der CIL vorliegt, auszuführen, wird jetzt die CLR benötigt. Diese „Common Language Runtime“ kann diesen CIL Code ausführen. Der Vorteil liegt dabei, ebenso wie in Java, darin, dass damit Programme, welche in CIL vorliegen, überall ausgeführt werden können, sofern eine CLR für die Plattform existiert.
[CVM]

3.3.3. Arbeitsweise der CLR

Die „Common Language Runtime“ arbeitet sehr ähnlich wie die JVM. Auch hier wird der Zwischencode, durch die CLR, in entsprechenden Maschinencode umgewandelt. Kleine Unterschiede gibt es lediglich in der Strategie der Umwandlung. Hier wird bei .NET im Normalfall das gesamte Programm, bei Programmstart, für die aktuelle Plattform in Maschinencode übersetzt. Es ist allerdings auch hier möglich einen JIT-Compiler zu verwenden.
[CVM]

3.3.4. Unterschiede zwischen Java und .NET

Auf den ersten Blick scheint es, dass sich Java und .NET in Hinsicht auf die virtuelle Maschine praktisch nicht unterscheiden. Das stimmt zwar für das grundsätzliche Konzept, im Detail unterscheiden sich die beiden virtuellen Maschinen JVM und CLR doch in gewissen Punkten.

Dies liegt vor allem daran, dass die Philosophie der beiden Technologien, in gewissen Bereichen, doch sehr voneinander abweicht.
So war beim Design von Java die Portabilität bzw. die Plattformunabhängigkeit ein vorrangiges Kriterium. Im Gegensatz dazu war für .NET die Performance ein sehr wichtiger Punkt. Das ist auch der Grund dafür, dass bei .NET auf die Interpretation der Zwischensprache gänzlich verzichtet wurde, während Java, vor allem zu Beginn, sehr intensiv mit Interpretation des Bytecodes gearbeitet hat. Das wirkt sich auch auf den Befehlssatz der jeweiligen virtuellen Maschine aus. Nähere Informationen dazu finden sie bei [CVM].

Ein weiterer Unterschied im Design ist, dass es bei der CLR von .NET ein wichtiges Ziel war, Support für mehrere Sprachen mitzubringen. Bei der JVM von Java war hingegen nicht geplant andere Sprachen, außer Java, zu unterstützen. Auch dieses Designziel hatte Auswirkungen auf die jeweilige virtuelle Maschine. So gibt es beispielsweise im Java Bytecode keine Unterstützung von Referenz Parametern, was es für sehr viele Sprachen schwierig macht, diese mit der JVM kompatibel zu machen.
[CVM]

3.4. VMWare

VMWare ist eine virtuelle Maschine für die x86 Architektur, mithilfe derer es möglich ist, mehrere Betriebssysteme gleichzeitig auf einem System laufen zu lassen.

3.4.1. Klassifikation

Bei der VMWare handelt es sich um eine System virtuelle Maschine (siehe Punkt 2.2). Konkret ist es eine klassische System virtuelle Maschine (siehe Punkt 2.4.1), also eine virtuelle Maschine, auf der das Gastsystem und die Hardware die gleiche ISA-Schnittstelle benutzen.
Dadurch kann VMWare sehr effizient arbeiten, weil es die Hauptaufgabe von VMWare ist, die bestehende Hardware auf mehrere Gastsysteme aufzuteilen und dabei keine Emulation anderer Hardware durchgeführt werden muss.

3.4.2. Arbeitsweise von VMWare

Es gibt 3 verschiedene virtuelle Maschinen von VMWare. Diese sind Workstation, GSX Server und ESX Server.

VMWare Workstation

VMWare stellt verschiedene virtuelle Rechner dar, auf denen dann Betriebssysteme und Anwendungsprogramme installiert werden können. Diese werden entweder innerhalb eines Fensters im gastgebenden Betriebssystem oder im Vollbildmodus angezeigt. In jedem Fall ist es aber jederzeit möglich, zwischen dem Gastgeber und dem Gast umzuschalten.
[VMW]

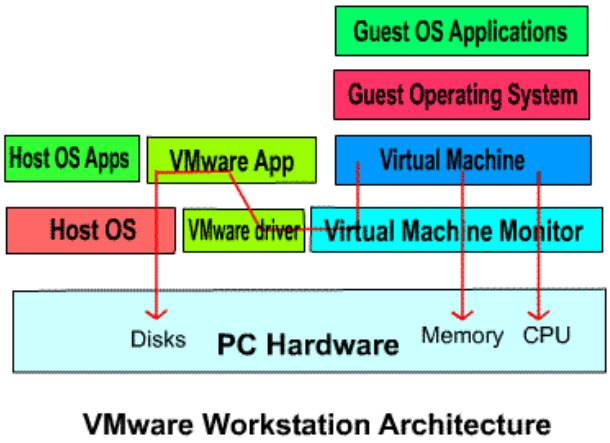

Abbildung 10: Die Architektur der VMWare Workstation [VMW]

VMWare Worstation besteht im Wesentlichen aus 3 Komponenten. Der „VMWare App“, dem „VMWare driver“ und dem „VMM“ (Virtual Machine Monitor).

Dabei erfolgen zugriffe auf den Hauptspeicher bzw. die CPU mittels dem VMM direkt auf die Hardware. Das hat den großen Vorteil, dass rechenintensive Aufgaben praktisch mit nativer Geschwindigkeit ausgeführt werden können und keine CPU emuliert werden muss. . Eine Einschränkung dadurch ist allerdings, dass nur Betriebssysteme mit der gleichen ISA-Schnittstelle wie die vorhandene Hardware installiert werden können.

Ist ein Zugriff auf ein I/O Gerät notwendig, so wird der Prozess auf dem Gastsystem unterbrochen und der Zugriff auf das Gerät wird durch einen Systemaufruf der VMWare App realisiert. Das Ergebnis des Zugriffs wird an den VMM zurückgegeben und der Prozess auf dem Gastsystem fortgesetzt. Für den Prozess auf dem Gastsystem sieht es so aus, als ob die Daten direkt von der Hardware kommen. Der Vorteil bei diesem Vorgehen liegt dabei daran, dass die Gerätetreiber des gastgebenden Systems genutzt werden können um die I/O Operationen auszuführen. Der Nachteil am Benutzen des gastgebenden Betriebssystems für die I/O Zugriffe ist die Tatsache, dass dadurch, speziell bei I/O lastigen Anwendungen, die CPU Auslastung steigt. Dies liegt daran, dass CPU Zeit verbraucht wird, um zwischen dem Gastsystem und dem gastgebenden System umzuschalten. Der CPU Overhead bei I/O Zugriffen wird also zwangsweise steigen.

Ein weiterer Nachteil der Implementierung von VMWare Workstation als gehostete virtuelle Maschine ist, dass alleine das gastgebende Betriebssystem die Kontrolle über die Hardwareressourcen hat. Das heißt, dass die virtuelle Maschine, wie jeder Prozess des

gastgebenden Betriebssystems, auf dessen Scheduling angewiesen ist. Das kann zu Performancenachteilen, im Vergleich zur ESX-Server Variante, führen.
[HVMA]

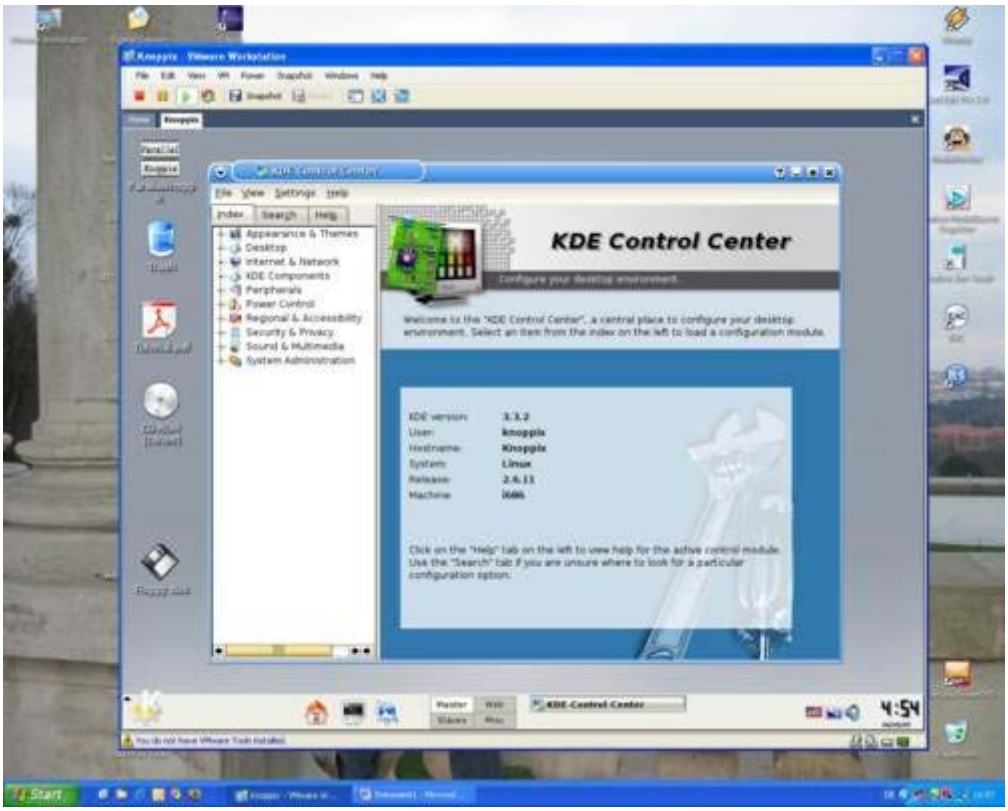

Abbildung 11: Knoppix, das in einem Fenster der VMWare Workstation unter Windows XP ausgeführt wird

VMWare GSX-Server

Die Architektur des VMWare GSX-Server entspricht im Wesentlichen jener der VMWare Workstation. Der VMWare GSX-Server bietet darüber hinaus Unterstützung von Mehrprozessorsystemen und bietet erweiterte Möglichkeiten für das Management.

VMWare ESX-Server

Die ESX Server Variante von VMWare ist hingegen eine virtuelle Maschine die direkt auf der Hardware läuft. Dies ist, aus Sicht der Performance, noch besser als die anderen Varianten von VMWare, weil auch hier direkt die Hardware benutzt wird, allerdings hat diese Variante den zusätzlichen Vorteil, dass durch das direkte Aufsetzen auf der Hardware auch Zugriffe auf I/O Geräte direkt erfolgen.

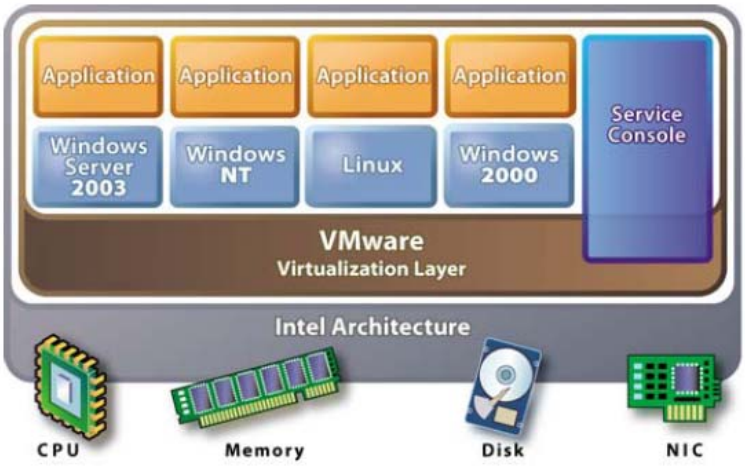

Abbildung 12: VMWare ESX-Server läuft direkt auf der Hardware[ESXAG]

Der VMWare ESX-Server läuft also direkt auf der Hardware und bildet damit eine Art minimales Betriebssystem. Dieses basiert auf einem Linux Kernel. Innerhalb der virtuellen Maschine können dann mehrere Betriebssysteme parallel installiert werden. Der VMWare ESX-Server stellt Tools zur Verfügung um Ressourcen an spezielle Gastsysteme zuweisen zu können bzw. die Ressourcennutzung einzuschränken wenn mehrere Gastsysteme gleichzeitig auf eine Ressource zugreifen.

Die Wartung des ESX-Server findet dabei nicht direkt über Tastatur und Monitor des Servers, sondern über ein Web-Interface mittels eines Client PCs, der mit dem Server verbunden ist, statt.
[ESX]

3.5. PearPC

PearPC ist eine virtuelle Maschine, welche die PowerPC Architektur auf einer x86 Architektur emuliert.

3.5.1. Klassifikation

Bei PearPC handelt es sich um eine System virtuelle Maschine (siehe Punkt 2.2). Konkret ist es ein Emulator (eine „Whole System Virtual Machine“ – siehe Punkt 2.4.2). Das heißt, dass die „virtualisierende Software“ hier als ein Adapter zwischen zwei unterschiedlichen Architekturen (mit 2 unterschiedlichen ISA-Schnittstellen) dient.
Das hat zur Folge, dass sämtliche Software, die in dieser virtuellen Maschine läuft, nicht direkt auf der vorhandenen Hardware ausgeführt werden kann, sondern die Hardware emuliert werden muss. Dadurch ist die Performance von PearPC ziemlich schlecht.

3.5.2. Arbeitsweise von PearPC

PearPC ist, wie bei Emulatoren üblich, eine gehostete virtuelle Maschine. Das heißt, dass die PearPC Software in einem Gastbetriebssystem installiert und ausgeführt wird. Anschließend kann die jeweilige Software innerhalb der virtuellen Maschine ausgeführt werden.

Dabei ist es bei PearPC nicht möglich, die vorhandene Hardware direkt zu verwenden, weil die Software, die innerhalb von PearPC ausgeführt werden kann für die PowerPC Architektur bestimmt ist und diese eine andere ISA-Schnittstelle als die x86 Architektur besitzt.

Deshalb ist es nötig, die gesamte Hardware eines PowerPC zu emulieren, was ein sehr zeitaufwendiger Prozess ist. Es ist dabei nötig, sämtliche Instruktionen, die innerhalb der virtuellen Maschine ausgeführt werden sollen, durch Instruktionen der real vorhandenen Architektur zu ersetzen, welche den gleichen Effekt erzielen.

4. Zusammenfassung

Virtuelle Maschinen sind ein gutes Mittel, um Kompatibilität zwischen verschiedenen Systemen zu erreichen. Es ist allerdings zu beachten, dass virtuelle Maschinen nur dann sehr performant arbeiten, wenn diese die Hardware möglichst direkt verwenden können. In den Beispielen aus diesem Paper trifft das vor allem auf VMWare, aber auch auf die virtuellen Maschinen für die Hochsprachen Java und .NET zu.

Bei VMWare wird das dadurch erreicht, dass die Gastsysteme, welche in der virtuellen Maschine laufen, für die gleiche Architektur ausgelegt sein müssen, die auch tatsächlich existiert. Dadurch können die Befehle, welche innerhalb der virtuellen Maschine aufgerufen werden, direkt an die Hardware weitergeleitet werden.
Bei Java und .NET wird hingegen ein anderer Ansatz gewählt, um auch auf unterschiedlichen Architekturen lauffähig zu sein. Hier werden die Programme, welche in diesen Sprachen geschrieben wurden, in abstrakten Maschinencode übersetzt. Die Arbeit der virtuellen Maschine besteht darin, diesen abstrakten Maschinencode in Maschinencode umzuwandeln, der auf der tatsächlich vorhandenen Hardware ausführbar ist. Da der abstrakte Maschinencode sehr allgemein gehalten ist, ist diese Umwandlung relativ einfach und muss auch nur einmal durchgeführt werden. Danach kann der echte Maschinencode für die Ausführung verwendet werden.

PearPC bietet hingegen nicht die Möglichkeit die Hardware direkt zu nutzen. PearPC muss Maschinencode, der für eine Architektur kompiliert wurde (PowerPC) auf einer gänzlich anderen Architektur (x86) zum laufen bringen. Dies hat zur Folge, dass die PowerPC Architektur praktisch gänzlich emuliert werden muss. Die daraus resultierende Performance ist, verglichen mit den anderen hier vorgestellten virtuellen Maschinen, eher schlecht. Deshalb ist PearPC wohl eher als Forschungsprojekt zu betrachten, als eine Möglichkeit Software für PowerPC auf einer x86 Architektur effektiv einzusetzen.

5. Literatur

[CVM] K. John Gough ; Stacking them up: a Comparison of Virtual Machines ; Proceedings of the Australian Computer Systems and Architecture Conference, February 2001.

[ESX] VMWare ESX-Server
Url: http://www.pcw.co.uk/itweek/software/2085877/vmware-esx-server, Stand 7.6.05

[ESXAG] VMWare ESX-Server Administration Guide
Url: http://www.vmware.com/pdf/esx25_admin.pdf, Stand 7.6.05

[HVMA] A Hosted Virtual Machine Architecture
Url: http://www.usenix.org/publications/library/proceedings/usenix01/sugerman/sugerman_html/node3.html, Stand 7.6.05

[JAN] Bart Van Rompaey, Java and .NET: A look into today's virtual machine technology. UNIVERSITEIT ANTWERPEN 2003-2004

[VMA] J. E. Smith and Ravi Nair, Virtual Machine Architectures, Implementations and Applications, Morgan Kaufmann Publishers, 2005

[VMW] Virtual Machines & VMWare, Part I.
Url: http://www.extremetech.com/article2/0,1558,10403,00.asp, Stand 8.5.05

[Wiki] Wikipedia, die freie Enzyklopädie – Schnittstelle
Url: http://de.wikipedia.org/wiki/Schnittstelle, Stand 2.6.05