Ansätze zur Virtualisierung von Rechnern

Christian Surauer
Technische Universität München

25. April 2002

Inhaltsverzeichnis

1.0 Einleitung

Heutzutage sind selbst Arbeitsplatzrechner relativ leistungsstark. Die zur Verfügung gestellte Rechenleistung wird von den verbreiteten Anwendungen kaum genutzt, so dass man problemlos mehrere Anwendungen parallel ausführen kann. Obwohl es für x86-PCs mehrere Betriebssysteme gibt, werden viele Anwendungen nur für ein spezielles Betriebssystem angeboten. Ein Anwendungswechsel kann so durchaus einen Betriebssystemwechsel nach sich ziehen. Ein Manko der x86-Architektur ist, dass sie nicht so konzipiert ist, dass sie mehrere gleichzeitig laufende Betriebssysteme unterstützt. Mit der Virtualisierung kann man erreichen, dass mehrere Betriebssysteme simultan benutzt werden können. Dadurch wird der Neustart des Rechners zum Systemwechsel unnötig. Außerdem können instabile Betriebssysteme getestet werden, während stabile zur Arbeit benutzt werden. Virtuelle PCs können auch als Debug-Werkzeug eingesetzt werden, um Fehler in Betriebssystemkernen zu finden. Weitere Ziele, die eine Virtualisierung der Hardware adressieren kann, sind besserer Schutz der Nutzer eines Mulitusersystems voneinander und der Betrieb von einfachen, verbreiteten Betriebssystemen auf skalierenden Multiprozessorsystemen. Bisher galt die Virtualisierung der x86-Hardware als sehr schwer oder sogar unmöglich. Mit dem Erscheinen des kommerziellen Produktes VMware wurde jedoch die Realisierbarkeit gezeigt.
Diese Arbeit führt in das Themengebiet der Emulation und Virtualisierung von Rechnern ein und diskutiert Techniken, die die typische Hardware virtualisieren.

2.0 Problemstellung

Anwendungen, die für verschiedene Betriebssysteme erstellt wurden, sollen gleichzeitig benutzt werden können.

2.1 Naiver Ansatz

Der naive Ansatz ist, einfach mehrere Betriebssysteme gleichzeitig auszuführen. Warum das nicht funktioniert, wird im Folgenden erläutert.
Die Hauptaufgaben eines Betriebssystems sind die Hardware zu steuern, Ressourcen zu verwalten und Anwendungen störungsfrei und geschützt voreinander parallel laufen zu lassen. Dabei haben Betriebssysteme im Allgemeinen exklusiven und vollständigen Zugriff auf den gesamten Rechner. Die parallele Ausführung von Betriebssystemen auf x86-Hardware bereitet Probleme. Zum Einen ist die x86-Architektur für derartige Aufgaben nicht ausgelegt, zum Anderen wurden im Laufe der Entwicklung der gängigen PC-Betriebssysteme keine Anstrengungen unternommen, Mechanismen zu schaffen, die es ermöglichen, BS kooperativ oder konkurrierend simultan auszuführen. Daraus ergibt sich, dass Betriebssysteme ohne Änderungen an deren Quelltext oder sogar ihrem Design nicht parallel auszuführen sind. Eine spezielle Software ist nötig, die in der Lage ist, mehrere Systeme gleichzeitig kontrolliert auszuführen.

2.2 Mögliche Lösungen

Es kommen grundsätzlich drei Verfahren in Frage:

- Emulation der Programmierschnittstelle des Betriebssystems (API),
- Emulation eines kompletten Rechners
- Virtualisierung des Rechners.

2.2.1 Emulation der API

In vielen Fällen ist der Anwender nur an den Applikationen interessiert, nicht am Betriebssystem selbst. Sollen also nur Anwendungsprogramme, keine weiteren Betriebssystemkerne laufen, bietet sich an, nur die API des zweiten Betriebssystems zu emulieren. Systemrufe werden übersetzt und eventuell nötige Bibliotheken bereitgestellt. Dadurch ist dieses Verfahren sehr schnell. Ein zweiter Vorteil ist, dass keine Kopie des zweiten Betriebssystems nötig ist. Ein Nachteil ist, dass für jede Kombination von Betriebssystemen eine API-Übersetzung existieren muss. Auch spezifisches Verhalten der API muss berücksichtigt werden (evtl. müssen Bugs der API emuliert werden). Solche Übersetzungen sind aufgrund der teilweise sehr umfangreichen und schlecht dokumentierten APIs nur mit großem Aufwand zu erstellen und man kann auf diese Weise keine Betriebssystemkerne testen, wie es für die Betriebssystementwicklung interessant ist.
Ein Beispiel ist WINE, das 1993 ins Leben gerufen wurde. Wine implementiert die Win32 API unter Intel Unix-Derivaten, wie Linux und Solaris. Das Programm-Packet enthält einen Program Loader, der es erlaubt, unmodifizierte Win32 Binaries auszuführen.

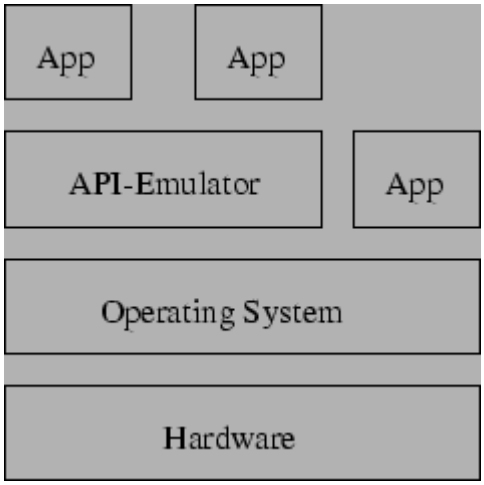

Abbildung 2.1: Emulation der API eines Betriebssystems

2.2.2 Emulation des Rechners

Bei der Emulation wird ein ganzer Rechner komplett in Software nachgebildet. Das heißt, CPU und Geräte sind Datenstrukturen, die von der Emulations-Software verändert werden. Zur Ein- und Ausgabe benutzt der Emulator den Wirt. Alle Assembler-Befehle werden von der Software geladen, dekodiert und die Wirkung auf die Datenstrukturen ausgeführt.
Ein Vorteil der Emulation ist, dass die emulierte Architektur eine ganz andere sein kann, als die, auf der die Emulation läuft. Außerdem werden auf der Emulation gesamte Betriebssysteme mit entsprechenden Anwendungen ausgeführt, was den Emulator zu einem Debug-Werkzeug für Betriebssystemkerne macht. Diesen Vorteilen steht der Nachteil gegenüber, dass das Verfahren der Emulation sehr langsam und aufwendig ist, da jeder Befehl geeignet emuliert werden muss.
Ein Vertreter dieser Art der Emulation ist Bochs. Bochs versteht sich selbst als portable x86-PC-Emulationssoftware, die genug von der x86 CPU, der Hardware und des BIOS emuliert, um DOS, Windows 95 und andere Betriebssysteme laufen zu lassen.

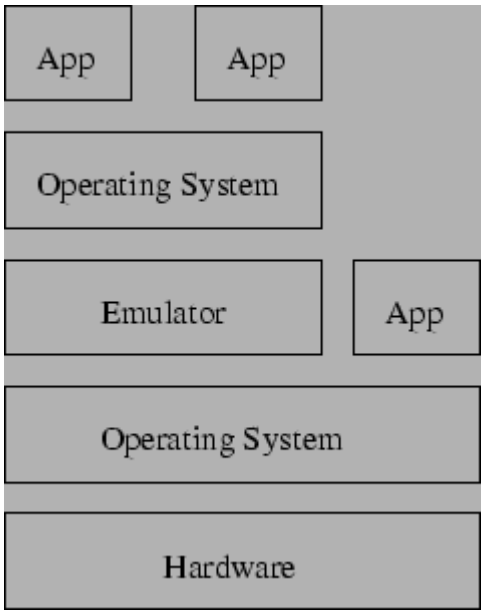

Abbildung 2.2: Emulation der gesamten Hardware

2.2.3 Virtualisierung des Rechners

Wenn ein Emulator die gleiche Hardware wie die nachahmt, auf der er läuft, liegt der Gedanke nahe, nicht alles in Software nachzubilden, sondern einen Teil der echten Hardware zu verwenden. Dies ist nicht uneingeschränkt möglich, denn es gibt gewisse Restriktionen. Die meisten Befehle können von der CPU unverändert ausgeführt werden, aber einige führen zu Kollisionen, wenn mehrere Betriebssysteme gleichzeitig aktiv sind. Diese kritischen Befehle betreffen vor allem die Ein- und Ausgabe sowie die Speicherverwaltung und die Schutzmechanismen. Sie dürfen nicht wirklich, sondern nur virtuell ausgeführt werden, wobei ein Virtual Machine Monitor (VMM) die Virtualisierung vornimmt. Er beobachtet die Programmausführung, findet die problematischen Befehle und emuliert sie auf geeignete Art und Weise. Mit anderen Worten, zwischen Betriebssystem und Hardware wird eine zusätzliche Schicht, der Virtual Machine Monitor, eingeschoben, der die virtualisierte Hardware bereitstellt. Der Virtual Machine Monitor muss im Kernmodus laufen, da er vollen Zugriff auf den Rechner benötigt, um Befehle abzufangen und geeignet emulieren zu können, Geräte zu steuern und um Systemstrukturen entsprechend präparieren zu können.
Grundsätzlich lassen sich zwei Virtual Machine Monitor – Architekturen beobachten, die im Folgenden beschrieben werden.

Gleichberechtigte Gäste

Der Virtual Machine Monitor verwaltet hier alle Systemressourcen und nur er läuft im Kernmodus. Alle laufenden Betriebssysteme (Gäste) stehen unter der Kontrolle des Virtual Machine Monitor, befinden sich im Nutzermodus und sind einander gleichberechtigt (Abbildung 2.3).
Hier sind zwei Modelle denkbar:
Statisches System: Beim Booten wird festgelegt, welche Betriebssysteme laufen sollen. Betriebssysteme können beendet, aber keine neuen hinzugenommen werden. Ein Ersetzen des einen durch ein anderes Betriebssystem ist noch denkbar.
Dynamisches System: Während der Laufzeit können beliebige Gäste neu hinzukommen. Die Konfiguration ändert sich dynamisch im Betrieb, es gibt Mechanismen zum Starten und Beenden von Gästen.

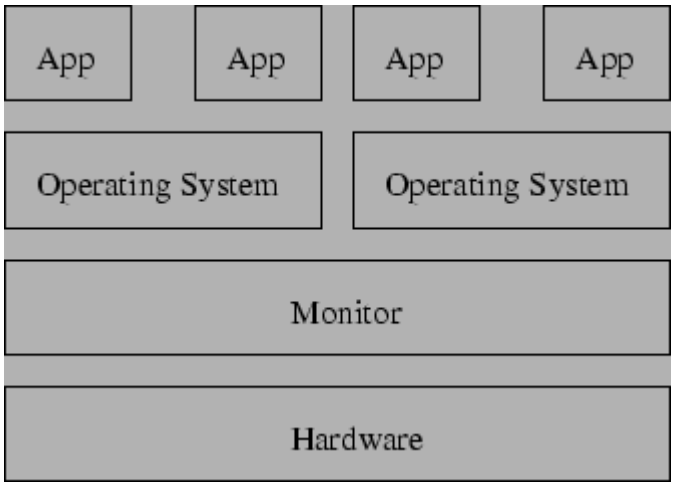

Abbildung 2.3: VMM ohne Wirt mit mehreren Gästen

Wirt-Gast-Struktur

Ein Betriebssystem (Wirt) läuft wie üblich auf einer realen Hardware. Der Virtual Machine Monitor läuft als Anwendungstask unter diesem Betriebssystem. Bekommt er Rechenzeit, sichert er den Kontext des Wirts und ersetzt diesen, so dass statt des Wirtes nun er im Kernmodus läuft. Die Gäste befinden sich auch hier im Nutzermodus. Ist die Zeitscheibe zu Ende, wird wieder ein Kontextwechsel hin zum Wirt ausgeführt. Abbildung 2.4 illustriert den Sachverhalt. Der Wirt verwaltet CPU, Hauptspeicher und sämtliche Geräte. Der Virtual Machine Monitor muss Ressourcen, die er dem Gast zur Verfügung stellt, beim Wirt reservieren. Das heißt, der Virtual Machine Monitor stellt einen Teil seiner Zeitscheibe, die ihm vom Wirt zugeteilt wird, dem Gast zur Verfügung. Außerdem muss Speicher, der den Hauptspeicher des Gastes bilden soll, beim Wirt allokiert werden. Gleiches gilt für Speicher, den der Virtual Machine Monitor für Verwaltungsinformationen benötigt. Der Gast kann Geräte nicht direkt ansprechen, es werden Treiber des Wirtes verwendet. Weitere Betriebssysteme werden wie neue Anwendungen gestartet und präsentieren sich ebenso dem Nutzer.

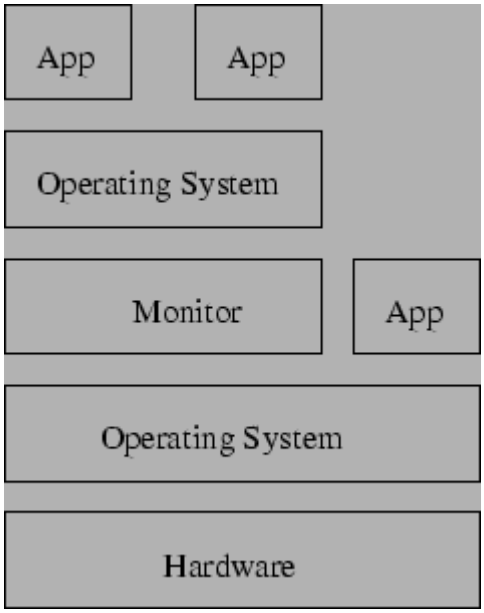

Abbildung 2.4: VMM mit Wirt und Gast

Gegenüber der Emulation, hat die Virtualisierung den Vorteil, dass eine höhere Geschwindigkeit erreicht wird. Auch die aufwendigen API – Emulationen fallen wie schon bei der Rechner – Emulation weg. Von Nachteil ist jedoch, dass die Performance virtualisierter Rechner gegenüber realer Rechner niedriger ist und die x86-PC-Hardware im Allgemeinen schwer zu virtualisieren ist.

Die Idee der Virtualisierung realer Hardware ist nicht neu. Bereits in den 60er Jahren beschäftigten sich Forscher mit der Virtualisierung der S/360-Hardware. Das primäre Ziel war, statt des Batchbetriebs ein "Multi User System" zu bauen. Die Rechenleistung eines Großrechners sollte mehreren Benutzern zur selben Zeit angeboten werden. Dazu ist mindestens virtueller Speicher nötig. Doch die Entwickler gingen gleich einen Schritt weiter und virtualisierten nicht nur den Hauptspeicher, sondern gleich die ganze Maschine. Hauptzweck war der Schutz der Nutzer voreinander. 1970 konnte das erste System mit einer virtuellen Maschine in Anwendung genommen werden. 1979 wurde das System unbenannt in VM/370, was damals auf IBMs Mainframes weit verbreitet war.
Das System wurde in zwei bedeutende Teile geteilt:
Der Virtual Machine Monitor (VMM), der auf der realen Hardware lauft. Er virtualisiert die Hardware so, dass mehrere virtuelle Maschinen auf eine reale Maschine abgebildet werden. Das Console Monitor System (CMS) stellt die eigentliche Betriebssystemfunktionalität zur Verfügung. CMS ist ein Single-User-System, und setzt auf dem Virtual Machine Monitor auf. Es existieren klare Schnittstellen zwischen Virtual Machine Monitor und CMS. Die Benutzung der Geräte durch CMS geschieht in spoolartiger Weise, das heißt die Aufträge werden in der Reihenfolge in der sie ankommen abgearbeitet.

Die x86-Hardware z.B. galt solange als nicht oder kaum virtualisierbar, bis es VMware gelang, mit einem verfügbaren, gebrauchsfertigen Produkt das Gegenteil zu zeigen. Es sind keine internen Verfahren oder Details bekannt. VMware soll als Beispiel dafür dienen, wie sich ein Monitor für virtuelle Maschinen in eine bestehende Umgebung eingliedern und dem Anwender präsentieren kann. Die Systemvoraussetzungen sind ein Pentium II mit 266 MHz und 96 MB RAM. Dies deutet darauf hin, dass die Virtualisierung eines x86-PCs ein ressourcenaufwändiges Problem ist. Bei der Installation werden neue virtuelle Geräte angelegt. Des Weiteren wird der Betriebssystemkern erweitert. Die Festplatte wird in Form einer Datei bereitgestellt, die im Dateisystem des Wirtes liegt.

Plex86, welches früher FreeMWare hieß, ist ein typisches OpenSource – Projekt. Es wird von einer internationalen Gemeinde entwickelt. Als Gründer leitet Kevin Lawton, welcher schon Hauptentwickler des PC-Emulators Bochs war, auch Plex86. Das Internet dient zur Kommunikation und Verbreitung des Quellcodes. Als Architektur wurde die Wirt-Gast-Struktur gewählt.

3.0 Abbildung virtueller auf reale Maschinen

In diesem Abschnitt wird näher auf den Virtual Machine Monitor der x86-Architektur eingegangen. Es werden Funktionen zur Abbildung virtueller auf reale Maschinen vorgestellt und diskutiert. Der Grundsatz lautet, soweit wie möglich die reale Maschine zu nutzen. Was auf diese Weise nicht ausgeführt werden kann oder darf, muss emuliert werden.

3.1 Abbildung virtueller CPUs auf eine reale CPU

Es gibt eine Vielzahl von Prozessor - Generationen und -Variationen. Da Betriebssysteme aus Gründen der Performance und der Genauigkeit an die Prozessoren gut angepasst sind, dürfen die Unterschiede nicht außer acht gelassen werden. Außerdem gibt es drei Betriebsmodi, den

Protected Mode, den Real Mode und den Virtual 86 Mode (V86). Für moderne Betriebssysteme ist nur der Protected Mode relevant. Die beiden anderen Modi werden nur noch aus Kompatibilitätsgründen mit sehr alten Betriebssystemen unterstützt. Daher liegt der Fokus in der Entwicklung hauptsächlich auf dem Protected Mode.

Die Befehle
Je nach Art der CPU (Hersteller, Prozessorgeneration) unterscheidet sich die Menge der Befehle. Für einen Virtual Machine Monitor sind folgende Gruppen von Befehlen zu unterscheiden:

- Unkritische Befehle
    - Nichtprivilegierte Befehle
- Kritische Befehle
    - Privilegierte Befehle
    - Nichtprivilegierte Befehle mit variablen Verhalten

In den folgenden Abschnitten wird auf diese Befehle näher eingegangen:

Unkritische Befehle
Die unkritischen Befehle stammen aus der Klasse der nichtprivilegierten Befehle. Diese Befehle verändern die Systemstrukturen der Maschine nicht und können dadurch von der CPU unmodifiziert ausgeführt werden. Ausnahmen bilden Befehle, deren Verhalten von der Privilegstufe abhängig ist, oder deren Verhalten nicht immer gleich ist. Dabei sollte man aber nicht vergessen, dass die meisten Befehle unkritisch sind, d.h. sie können von der CPU unverändert ausgeführt werden. Die Geschwindigkeit entspricht dabei der originalen Maschine. Daraus ergibt sich der signifikante Performancevorteil gegenüber Emulatoren. ADD oder MULT spiegeln diese Klasse der Befehle wieder. Es werden keine Systemzustände verändert und ihr Verhalten ist nicht von der Privilegstufe abhängig, in der sie ausgeführt werden.

Kritische Befehle
Privilegierte Befehle müssen abgefangen und emuliert werden, denn sie beeinflussen den Zustand der Maschine oft in einer für den Virtual Machine Monitor kritischen Weise. Ein Beispiel ist das Laden des Page Directory Base Register (PDBR), womit der Virtual Machine Monitor die Kontrolle über den Hauptspeicher verlieren würde. Dem Gast darf also zu keiner Zeit die Ausführung solcher Befehle gestattet werden. Ermöglicht werden kann dies, indem der Gast im Nutzermodus, also in einer Privilegierungsstufe größer 0 ausgeführt wird. Falls versucht wird, einen privilegierten Befehl im Nutzermodus auszuführen, unterbricht die CPU die Programmausführung durch eine Protection Exception und führt die Ausnahmebehandlung aus. Diese Ausnahmebehandlung kann ein Stück Monitorcode sein, der den entsprechenden Befehl emuliert. Nach erfolgter Emulation wird mit dem nächsten Befehl im Gast fortgefahren.
Neben den privilegierten Befehlen müssen auch die nichtprivilegierten Befehle mit variablen Verhalten abgefangen werden. Das Verhalten dieser Befehle hängt meist von dem Ring ab, in dem sie ausgeführt werden. Da der Gast statt in Ring 0 nun auf der Privilegstufe ungleich 0 läuft, verhalten sich diese Befehle anders, als es der Gast erwartet. Ein Beispiel ist der VERR Befehl, der überprüft, ob das spezifizierte Segment von der aktuellen Privilegstufe aus gelesen werden darf. Da sich das Gast-Betriebssystem jetzt aber in einer Privilegstufe ungleich 0 befindet, wird ein falsches Ergebnis zurückgegeben. Durch geeignete Methoden müssen solche Befehle vor ihrer Ausführung gefunden und emuliert werden. Hierbei hilft

jedoch nicht wie bei den privilegierten Befehlen die CPU, das Filtern muss mit Software erledigt werden. Entsprechende Verfahren heißen Scan Before Execution (SBE) oder Prescan. Abbildung 3.1 verdeutlicht, dass statt des Befehls ein Sprung in die Emulationsbibliothek stattfindet, wo sich die Routinen zur Emulation von Befehlen befinden. Nach der Emulation wird der Gast hinter dem emuliertem Befehl fortgesetzt.

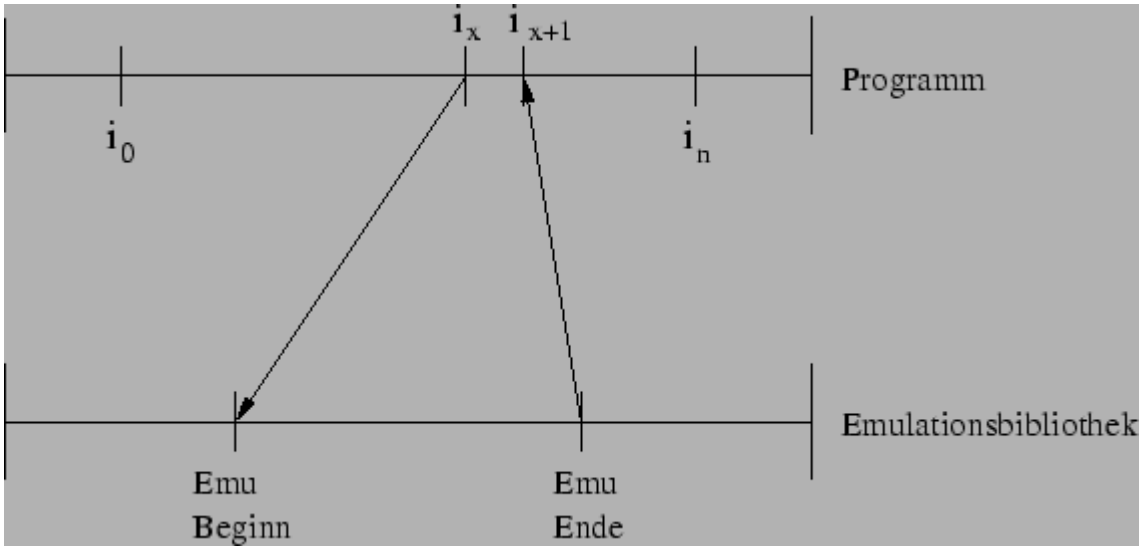

Abbildung 3.1: Erzwungene Emulation eines Befehls

Hier kommen einige Faktoren erschwerend hinzu:

- Das Befehlsformat der IA32-CISC-Architektur ist kompliziert.
- Es ist überlappender Code möglich, d.h. ein Befehl kann teilweise oder ganz ein Teil eines anderen Befehls sein.
- Der Code kann während der Laufzeit modifiziert werden (Self Modifying Code).

Das Auffinden dieser Befehle ist durch die lineare Suche in größeren Blöcken möglich. Die Befehle eines Blockes werden von vorn nach hinten untersucht, bevor der Gast den Code dieses Blockes ausführt. Ein Block kann z.B. eine Speicherseite oder sogar der komplette Betriebssystemkern sein. Sich überlappender Code kann aber nur schwer oder nicht gefunden werden, denn Befehle können an einer beliebigen Stelle im Speicher beginnen.
Eine andere Methode diese Befehle zu finden ist, den Ausführungspfad des Gastes zu verfolgen. An bestimmten Punkten, meist vor der Ausführung eines der hier betrachteten Befehls, wird der Gast gestoppt, der Befehl wird emuliert und es wird der Pfad, den der Gast beim Fortsetzen beschreitet, nach weiteren kritischen Befehlen abgesucht. An bedingten Sprüngen kann sich der Pfad verzweigen, so dass mehrere Stellen mit Haltepunkten versehen werden müssen. Jeder Befehl muss dabei von einem Softwaredecoder behandelt werden. Während dieses Verfahren aufwendiger ist, funktioniert es aber auch noch bei sich überlagernden Befehlen. Der Code wird also auf dem Pfad verfolgt, den der Gast ausführen würde. Das ist im Allgemeinen bis zum nächsten Sprung möglich. Falls es ein fester oder bedingter Sprung mit feststehender Zieladresse ist, kann man auch über den Sprung hinaus den Pfad verfolgen. Andernfalls muss man den Sprung emulieren, um den nächsten auszuführenden Befehl zu finden. Gelangt man bei der Pfadverfolgung an einen bedingten Sprung, kann man beide Pfade verfolgen und so rekursiv absteigen. Abbruchbedingungen für den Abstieg sind:

- Übertreten der Grenzen einer Speicherseite
- Eine maximale Tiefe ist erreicht

- Erreichen von schon gescanntem Code
- Erreichen von Befehlen, die Virtualisierung verlangen

Findet man nun Instruktionen, die einer Emulation bedürfen, aber keine Exception auslösen, muss man durch geeignete Maßnahmen die Emulation erzwingen. Folgende Verfahren kann man in Betracht ziehen:

- Hardwarebreakpoints die nur in begrenzter Anzahl zur Verfügung stehen und deshalb für diesen Zweck ungeeignet sind.
- Softwarebreakpoints; dafür stellt der Befehlssatz eine INT 3 genannte Instruktion zur Verfügung. Der Breakpoint ruft eine Exception auf, die der Virtual Machine Monitor verarbeiten kann.
- Ungültiger Opcode; beim Versuch, diese Operation auszuführen, wird eine entsprechende Exception ausgelöst, der Monitor kann reagieren.
- Sprung in Emulationsbibliothek; dabei wird der zu emulierende Befehl durch einen Sprung direkt zu der Routine ersetzt, die die Emulation durchführt; dabei befindet sich die Emulationsbibliothek entweder im Adressraum des Gastes, oder es werden schnellere Gates benutzt.

Allen vier in Frage kommenden Möglichkeiten ist gemein, dass sie Änderungen am Code des Gastes vornehmen müssen. Im Falle eines Softwarebreakpoints ist es nur ein Byte, bei einem Sprung sind es mehrere Bytes. Da man sich nicht darauf verlassen kann, dass ungültige Operationscodes auch ungültig bleiben und INT 3 gegenüber einem Sprung den Vorteil hat, nur ein Byte ersetzen zu müssen, ist die Verwendung von Softwarebreakpoints die eleganteste Methode.

Register

Die x86- Architektur ist so konstruiert, dass ein Befehl, der im Nutzermodus ausgeführt wird, eine Protection Exception hervorruft, wenn in ein Register geschrieben wird, das den Systemzustand beeinflusst. Somit ist eine einfache Kontrolle über die Systemregister möglich. Vom Gast gewollte Inhalte dieser Register müssen gesichert werden, damit sie beim Lesen zurückgeliefert werden können.

Das Lesen dieser Register ist hingegen problematischer, denn in einigen Fällen dürfen Systemregister von nichtprivilegierten Programmen ausgelesen werden. Liest aber der Gast aus, so müssen ihm die gesicherten Systemregister zur Verfügung gestellt werden. Es muss eine Emulation erzwungen werden.

Je nach Bedeutung der Register ist wie folgt mit ihnen zu verfahren:

Die Speicherverwaltungsregister TR, LDTR, IDTR, GDTR, wie auch die Steuerregister CR0 - CR4 beeinflussen den Zustand der Maschine in kritischer Weise und dürfen nicht mit den Werten belegt werden, die der Gast vorsieht. Auch das Flagregister EFlags und die Testregister TR3, TR4, TR5, TR6 und TR7 haben direkten Einfluss auf den Zustand der Maschine.

Vielzweckregister EAX, EBX, ECX und EDX, Befehlszeiger EIP, Stackzeiger ESP, Debugregister DR0 - DR7 und Segmentregister CS, DS, ES, FS, GS sind unkritisch für den Systemzustand, der Gast kann sie mit seinen Werten belegen.

3.2 Abbildung des virtuellen RAM auf realen RAM

Eine der ersten Entscheidungen, die man treffen muss, ist, ob man den verfügbaren Hauptspeicher in mehrere Partitionen für Wirt und Gäste teilt oder nicht. Damit entscheidet man sich auch für oder gegen Flexibilität in dieser Beziehung. Für eine Partitionierung spricht, dass die Speicherverwaltung für den Virtual Machine Monitor einfacher wird und sich DMA-Transfers einfacher oder überhaupt erst virtualisieren lassen. Dagegen spricht, dass man sich schon beim Booten für eine feste (Höchst-)Anzahl virtueller Maschinen entscheiden muss und dass ungenutzter Speicher nicht von anderen Gästen oder dem Wirt verwendet werden kann. Bei der Virtualisierung des Hauptspeichers kann auf Mechanismen zurückgegriffen werden, die schon von der Hardware für diesen Zweck vorgesehen sind, denn der Hauptspeicher ist eine Komponente des PCs, dessen Virtualisierung vorgesehen ist und unterstützt wird. Besagte Mechanismen sind Segmentierung und Paging. Die Virtualisierung des RAMs führt direkt zur Virtualisierung der Systemstrukturen speziell der Seitentabellen. In der x86-Architektur bestehen Seitentabellen aus einzelnen Seitenverzeichnissen, die jeweils 1024 Tabelleneinträge aufweisen. Jeder dieser Einträge hat sieben Komponenten:

- Page Frame Address - physische Adresse des Seitenrahmens
- Available - für das Betriebssystem verfügbar
- Dirty - auf diese Seite wurde (noch nicht) schreibend zugegriffen
- Accessed - auf diese Seite wurde (noch nicht) zugegriffen
- User/Superuser - auf diese Seite darf in allen Schutzstufen/nur von Ring 0 aus zugegriffen werden
- Read/Write - nur lesen bzw. lesen und schreiben ist erlaubt
- Present - diese Seite befindet sich (nicht) im Speicher.

Der Virtual Machine Monitor muss sicherstellen, dass nur die von ihm vorgesehenen Seitenrahmen in die aktiven Seitentabellen eingetragen werden. Die grundlegende Idee ist, für diesen Zweck eine dritte Stufe der Adressübersetzung einzuführen, wie es in Bild 3.2 dargestellt wird.

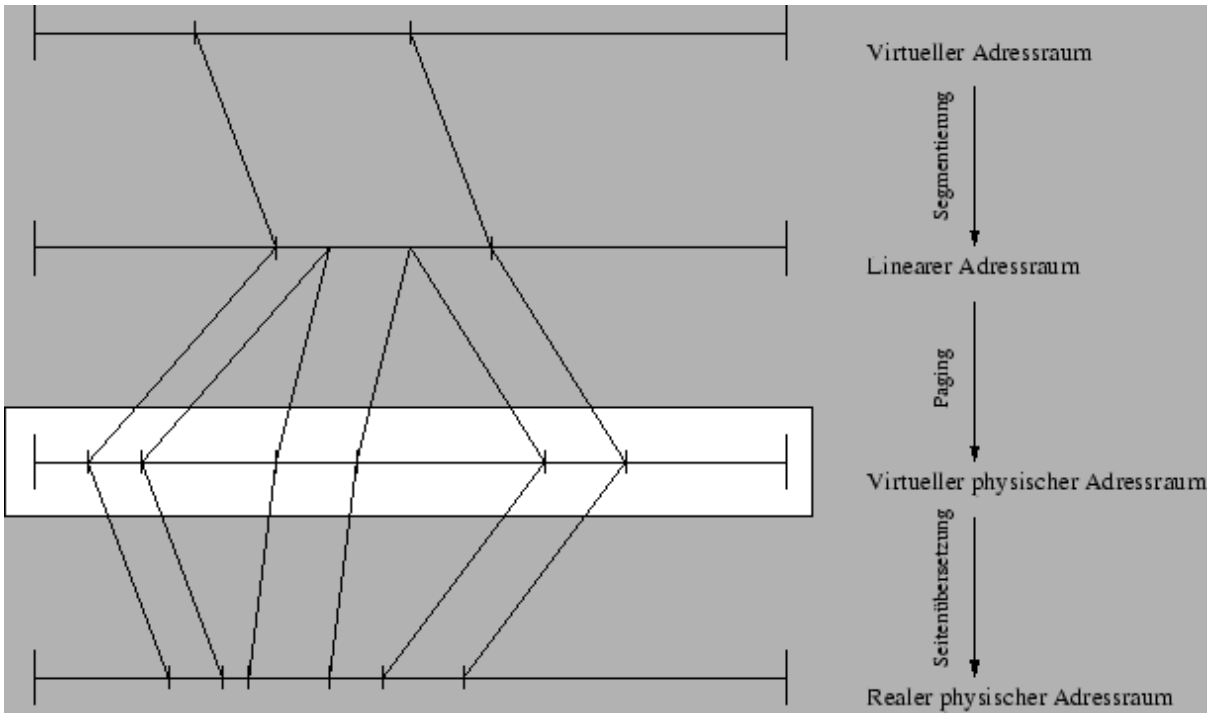

Abbildung 3.2: Einführung eines virtuellen physischen Adressraums

Der virtuelle Adressraum des Gastes wird wie üblich mit Hilfe der Segmente in einen linearen Adressraum überführt. Das Paging übersetzt die linearen Adressen in physische Adressen. Da diese jedoch virtualisiert werden, sind es nur aus Sicht des Gastes physische Adressen, sie bilden einen Virtuellen Physischen Adressraum, der vom Monitor gebildet wird. Im Bild ist er weiß hinterlegt. Um die Adressen aus dem virtuellen physischen Adressraum in den realen physischen Adressraum zu übersetzen, muss der Monitor eine dritte Stufe der Adressübersetzung bereitstellen, denn die Hardware unterstützt nur eine zweistufige Adressübersetzung.
Für die dritte Stufe existiert eine Tabelle, in der jede Seitenrahmenadresse des virtuellen Gasthauptspeichers eine reale physische Seitenrahmenadresse zugeordnet ist. Abbildung 3.3 zeigt, wie beim Kopieren die Adressübersetzung stattfindet.

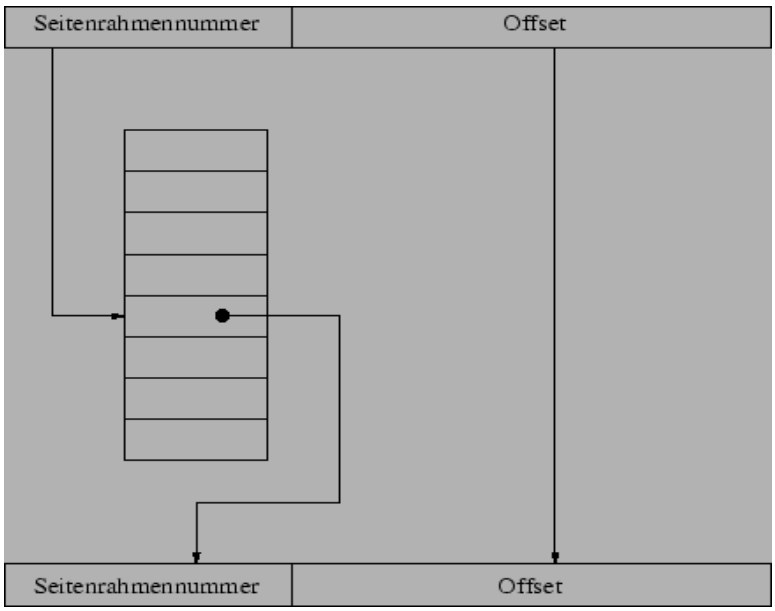

Abbildung: 3.3: Stufe der Adressübersetzung

Eine virtuelle physische Adresse besteht aus einer Seitenrahmennummer und einem Offset in der angegebenen Seite. Der Offset wird übernommen. Die virtuelle physische Seitenrahmennummer wird als Index in der genannten Tabelle interpretiert. Zu einer virtuellen Rahmennummer wird mit dieser Strategie leicht die korrespondierende reale Rahmennummer gefunden. Die neue Adresse setzt sich also aus der korrespondierenden Rahmennummer und dem alten Offset zusammen. Jeder Gast besitzt seine eigene Tabelle und hat somit nur Zugriff auf einen partitionierten Bereich des Hauptspeichers. Außerdem arbeitet die Zuordnung verschiedener virtueller Adressen zu derselben physischen Adresse genau wie vom Gast vorgesehen.
Ein weiterer Fakt, auf den man sein Augenmerk richten sollte, sind Superpages. Da ein Wirt nur selten entsprechende zusammenhängende Bereiche zur Verfügung stellen kann, muss eine große Seite aus 1024 kleinen Seiten zusammengesetzt werden. Im aktiven Page Deskriptor steht somit kein Verweis auf einen 4-MB-Seitenrahmen, sondern auf eine Page Table, in der wiederum sich die Verweise auf die kleinen Seitenrahmen befinden. Entsprechend werden auch die Available- und Dirty- Bits gehandhabt. Sie ergeben sich aus einer bitweisen ODER-Operation aller einzelnen A- und D- Bits. Mit der Zerlegung einer Superpage in kleine Seiten geht jedoch ihr Vorteil verloren, nämlich die größere Geschwindigkeit, mit der lineare auf physische Adressen abgebildet werden.
Wird nicht mit Speicherpartitionierung gearbeitet, sondern Wert auf große Flexibilität gelegt, ist eine dynamische Seitenverwaltung mit einem Seitencache sinnvoll. Dabei werden Wirt und Gästen Speicherseiten während der Laufzeit entzogen und hinzugefügt. Zum Preis eines zusätzlichen Verwaltungsaufwandes kann freier Speicher immer dem Betriebssystem zugeführt werden, das ihn gerade benötigt.

3.3 Abbildung der virtuellen Systemstrukturen auf reale Systemstrukturen

Relevante Systemstrukturen sind die Seitentabelle (Page Table, PT), die oben schon behandelt wurde, die Globale Deskriptortabelle (GDT), die Lokale Deskriptortabelle (LDT) und die Interruptdeskriptortabelle (IDT).
Wenn die Systemstrukturen genau so übernommen würden, wie es der Gast vorsieht, würde er mehr Kontrolle über den Rechner bekommen, als für den stabilen Betrieb möglich ist.
Ein Beispiel: Könnte der Gast Seitentabellen nach eigenen Vorstellungen setzen, könnte er an beliebigen Stellen des Hauptspeichers schreiben. Weder der Virtual Machine Monitor, noch andere Gäste oder der Wirt würden lange laufen, da ihre Daten und ihr Code unerwartet verändert würde. Die Gast-Systemstrukturen können also nur modifiziert verwendet werden.

3.4 Abbildung von virtuellen Geräten auf reale Geräte

Üblicherweise werden in allen Rechnern einige Geräte eingebaut. Sie lassen sich charakterisieren durch:

- Interrupts
- I/O-Ports
- Bereiche im Adressraum des Prozessors für Memory Mapped IO
- Die Fähigkeit, direkt (ohne die CPU in Anspruch zu nehmen) auf den Hauptspeicher des Rechners zuzugreifen (Direct Memmory Access)
- Protokolle, mit dem das Gerät mit seinem Treiber kommuniziert
- Innere Zustände, in denen sich das Gerät befinden kann

Ohne Geräte zur Ein- und Ausgabe bleibt der virtuelle Rechner für den Anwender unbenutzbar. Zur Virtualisierung sind mehrere Architekturen möglich, die in diesem Abschnitt diskutiert werden.
Die Wahl der passenden Architektur hängt vom Zweck ab, den man verfolgt. Kommt es auf eine möglichst genaue Virtualisierung an, muss dem Gast das reale Gerät virtuell zur Verfügung stehen. In Testsituationen und Realzeitanforderungen kann andererseits aber auch verlangt werden, dass ein Gast direkten Zugriff auf das Gerät bekommt, was entsprechende Konsequenzen bezüglich der Sicherheit nach sich zieht.

Ports
Ports sind spezielle Adressen im Adressraum des Prozessors. Alle diese Adressen zusammen werden als I/O-Adressraum bezeichnet. Auf die Ports wird mit besonderen Befehlen zugegriffen: in, out, ins und outs. Die Adresse des Ports ist im Befehlswort codiert oder steht in einem Register. In einer Bitmap kann das Betriebssystem angeben, welche Ports für im Nutzermodus laufende Programme offen sind oder gesperrt werden sollen. Sperrt man alle Ports, führt deren Benutzung zur Protection Exception. Der Monitor kann so die Kontrolle übernehmen. Eine zweite Möglichkeit, den Zugriff auf I/O-Adressen zu erlauben bzw. zu verbieten, eröffnen die EFlags. Hier kann die Privilegstufe angegeben werden, die mindestens erforderlich ist, um auf die Ports zuzugreifen.

Eingeblendeter Gerätespeicher
Einige Geräte, zum Beispiel Grafikkarten, blenden den in ihnen eingebauten Speicher in den Adressraum der CPU ein. Hier bietet sich an, statt des Speichers im Gerät normalen Hauptspeicher zu benutzen und dann je nach Anwendungsfall von Zeit zu Zeit oder auf bestimmte Ereignisse hin ins Gerät zu kopieren. Ein Problem kann die Größe des Geräte-Speichers sein, der durchaus die Größe des eingebauten RAM erreichen kann. Ein Auslagern auf Hintergrundspeicher kommt aus Performancegründen nicht in Frage, somit kann den Gästen in besonderen Fällen nicht der volle Umfang des Speichers auf dem Gerät zur Verfügung gestellt werden.

Hardwareinterrupts
Tritt ein Interrupt auf, unterbricht die CPU den aktuell laufende Task, schaut in der Interrupt Descriptor Table (IDT) nach und springt zur angegebenen Adresse. Die Interruptdeskriptortabelle nimmt maximal 256 Deskriptoren für Task Gates, Interrupt Gates und Trap Gates auf. In der aktiven IDT finden sich Einträge für:

- Ausnahmebehandlung - deren Auswertung geschieht teilweise durch den Monitor
- Hardwareinterrupts - eine Behandlung wird gemäß dem Modell zur Virtualisierung der Geräte durchgeführt
- Softwareinterrupts - sie werden dem aktiven Gast zugestellt.

Um dem Gast die Interrupts zuzustellen, wird dessen virtuelle IDT verwendet. Je nach Virtualisierungsmodell wird der Interrupt direkt dem Wirt, direkt dem Gast oder situationsabhängig zugestellt. Im zweiten ist es nötig, dass der Virtual Machine Monitor dem Gast einen Interrupt zustellt, also einen Hardwareinterrupt emuliert. Dazu wertet er die vom Gast gesetzte virtuelle IDT aus und verfährt entsprechend.

Direct Memory Access (DMA)
Der Prozessor ist nicht der einzige Bestandteil eines Rechners, der direkt auf den Hauptspeicher zugreifen kann. Schon in den ersten PCs wurden Hilfsbausteine eingebaut, die Daten vom RAM zu den Geräten oder umgekehrt, sowie zwischen verschiedenen Plätzen im

Hauptspeicher selbstständig transferieren konnten. Da diese Hilfsbausteine nicht weiterentwickelt wurden, war die CPU beim Transfer bald schneller und sie wurden immer seltener verwendet. Andererseits kamen aber Geräte auf, die so genanntes Busmastering beherrschen. Auch dabei werden ohne den Prozessor in oder aus dem Hauptspeicher Daten transferiert. Gemein ist beiden Methoden, dass sie immer mit physischen Adressen arbeiten und keine Rücksicht auf die Speicherschutzmechanismen des Protected Mode nehmen. Einen Gast ungehindert DMA ausführen zu lassen würde also für den Monitor bedeuten, die Kontrolle zu verlieren, da der Gast nunmehr auf jeden beliebigen Speicherbereich zugreifen kann.

Protokoll
Das Protokoll, mit dem ein Gerät kommuniziert, ist für jedes Gerät spezifisch. Daher gibt es viele Protokolle, was den Aufwand bei einigen Modellen für virtuelle Geräte erheblich steigert.

Innerer Zustand
Viele Geräte besitzen einen inneren Zustand. Manchmal kann dieser Zustand abgefragt werden, aber nicht immer. Daher betrachtet man Geräte als Zustandsautomat, sofern es die Gerätevirtualisierungsarchitektur erfordert. Es wird angenommen, dass es sichere Zustände gibt, in denen ein Wechsel von einem Treiber auf einen anderen übergegangen werden kann. In unsicheren Zuständen darf nur ein einziger Treiber ein bestimmtes Gerät benutzen. Am Beispiel eines Modems ist "aufgelegt" ein sicherer Zustand. Ein beliebiger Treiber kann jetzt eine Nummer wählen und Daten senden bzw. empfangen. Während dieser Zeit darf kein anderer Treiber das Modem umprogrammieren, bis wieder "aufgelegt" wurde.

Mögliche Architekturen

Partitionierung der Geräte
Die Geräte werden in Partitionen eingeteilt, was durch die Partitionierung der Ports und Interrupts erreicht wird. Jedes Gerät wird von nur einem Treiber angesprochen. Das Gast-Betriebssystem kommuniziert direkt mit diesem Treiber. Der Nachteil ist allerdings, dass ein Gerät höchstens einem Betriebssystem zur Verfügung steht, was besonders kritisch bei nur einmal vorhandenen Geräten wie der Tastatur ist. Außerdem muss darauf vertraut werden, dass der Gast das Gerät korrekt steuert, denn der Monitor hat hier keine Möglichkeiten der Kontrolle.

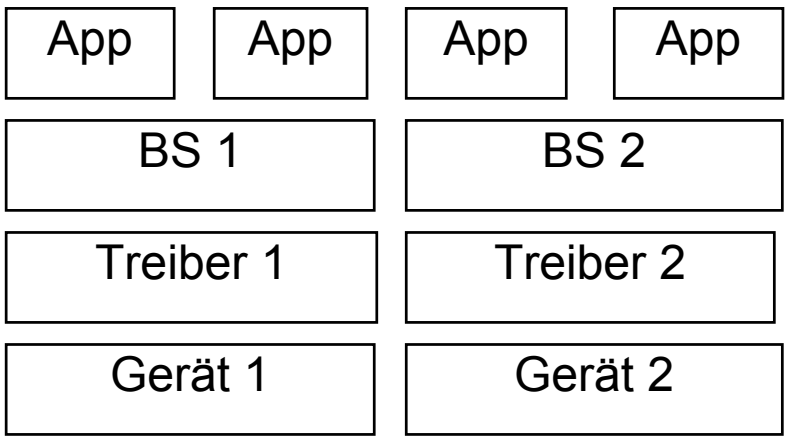

Abbildung: Partitionierung der Geräte

Mehrere Treiber pro Gerät
Ein Gerät wird in dieser Architektur von mehreren Treibern gesteuert. Da diese Treiber direkten Zugriff auf eingeblendete Gerätespeicher und Ports benötigen, können die Geräte schnell in inkonsistente Zustände kommen. D.h. die beiden Treiber müssen miteinander kommunizieren, um sich abzustimmen und auszuhandeln, wer den nächsten Hardware - Interrupt bekommt. Da diese Anforderungen in den Treibern der gängigen Betriebssysteme im Allgemeinen nicht implementiert sind, müssten diese Treiber entsprechend geändert werden, was enormen Aufwand bedeuten würde.

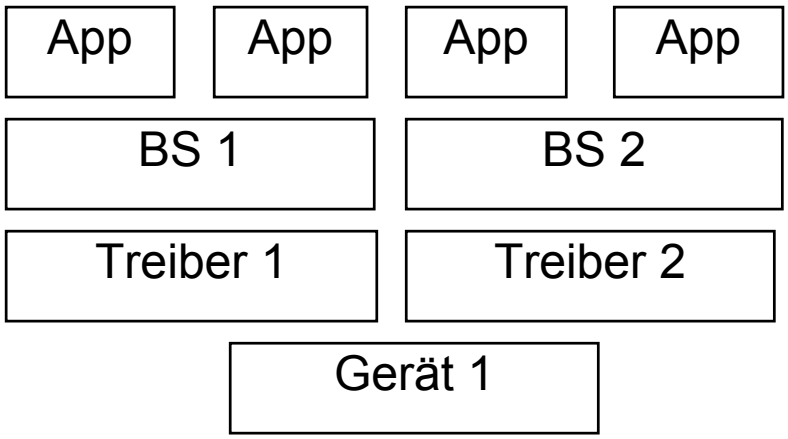

Abbildung: Zwei Treiber pro Gerät

Ein Multiplexer für mehrere Treiber
Im Gegensatz zum eben erläuterten Verfahren koordinieren sich die Treiber nicht selbst, sondern ein Multiplexer (MUX) übernimmt diese Aufgabe. Dafür muss der MUX die Geräte mit ihren Zuständen, der Kommunikation, Protokolle, usw. kennen. Außerdem muss sichergestellt sein, dass die Treiber keinen direkten Zugriff auf die einzelnen Gerät haben. Der Multiplexer kann die Zustände der Geräte in Software als Zustandsmaschinen nachbilden. Damit ist er in der Lage, das Gerät bei Bedarf in den Zustand zu versetzen, den ein bestimmter Treiber annimmt. Für Ausgabe-Geräte, die keine Echtzeitanforderungen haben, ist ein Spooling – Mechanismus denkbar, der die Geräte virtuell immer verfügbar macht und die Jobs dann aber nacheinander erledigt (Drucker – Warteschlange). Diese Anordnung ist ideal geeignet für die Architektur ohne Wirt mit gleichberechtigten Gästen. Alle Treiber kommunizieren hier mit den virtuellen Geräten, die der MUX jederzeit für jeden Gast zur Verfügung stellt. Allerdings muss der MUX für jedes der vielen möglichen Geräte implementiert werden, was einen sehr hohen Aufwand bedeutet Der VM/370 von IBM z.B. virtualisiert seine Geräte auch über einen Multiplexer.

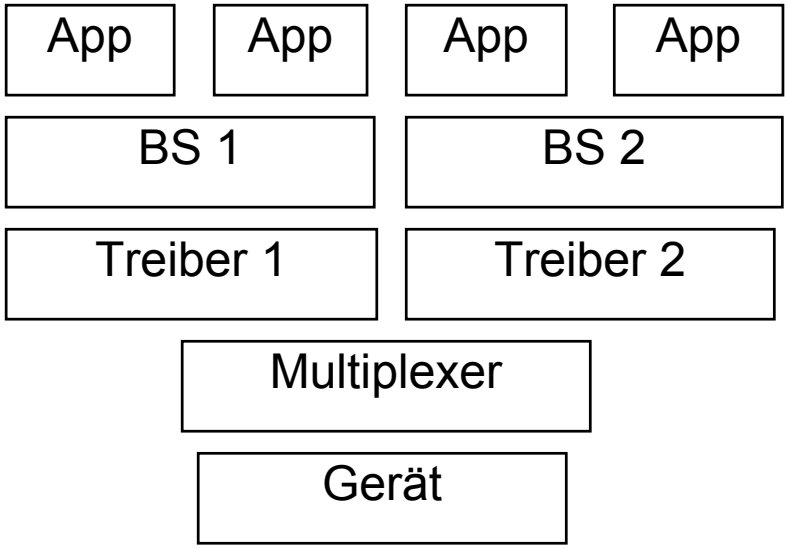

Abbildung: Ein Multiplexer für mehrere Treiber

Ein Pseudogerät emulieren und auf die Schnittstelle eines Treibers übersetzen
Hier läuft ein Betriebssystem wie gewöhnlich. Für jedes weitere Betriebssystem werden dazugehörige Pseudogeräte mit dazugehörigen Treibern angelegt. Ein Übersetzer, der zwischen den virtuellem Gast-Treiber und dem Wirt-Treibers platziert wird, sorgt für die Kommunikations-Verbindung. Wie die Terminologie schon andeutet, ist dies eine ideale Architektur mit Wirt und Gästen. Jedoch muss für jedes Gerät ein Übersetzer für die Schnittstellen der virtuellen Gast-Treiber und dem Wirt-Treiber geschrieben werden, was einen hohen Aufwand bedeutet. Man kann sich aber auf die Virtualisierung eines Pseudo-Gerätes beschränken, um den Aufwand zu senken. Ein großer Vorteil ist, das die direkte Kommunikation mit dem Gerät nur ein Treiber vornimmt, der allen weiteren Betriebssystemen zur Verfügung steht.

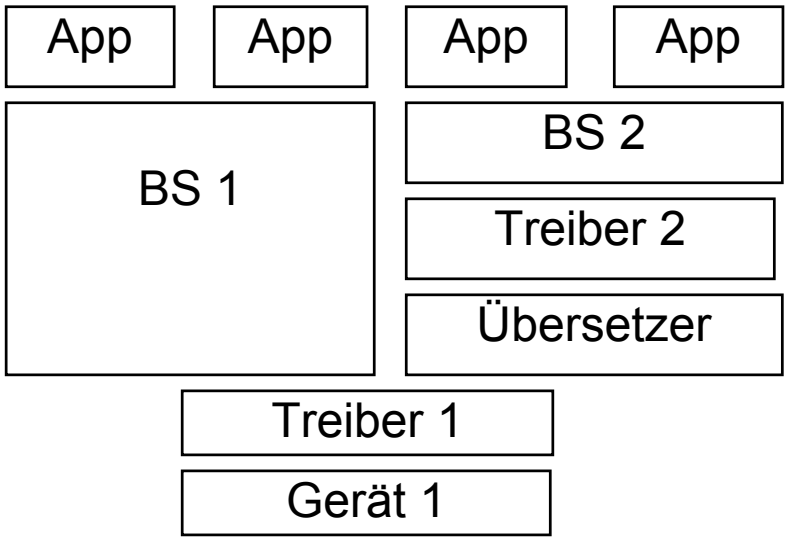

Abbildung: Emulation eines Pseudogerätes

Mehrere Betriebssysteme nutzen denselben Treiber

Es gibt nur noch einen Treiber pro Gerät, den mehrere Betriebssysteme nutzen. Dieser Aufbau würde nur dann funktionieren, wenn es gelingt, den Treiber in beiden Betriebssystemen sichtbar zu machen. Jedoch sind die praktischen Hindernisse zu hoch. Beide Betriebssysteme müssten mit diesem Treiber arbeiten können (zwei gleichartige Betriebssysteme), außerdem kann man meist nur schwer abgrenzen, welcher Code und welche Daten zu diesem Treiber gehören. Nicht zuletzt ist auch hier auf das Konsistenzproblem bei konkurrierendem Zugriff auf ein Gerät zu achten. Der Treiber muss also auch die Aufgabe eines Multiplexers übernehmen. Um diese Architektur verwenden zu können, wird man um einen Pseudotreiber für ein Betriebssystem nicht herum kommen, der Anfragen auf die Schnittstelle des realen Treibers übersetzt.

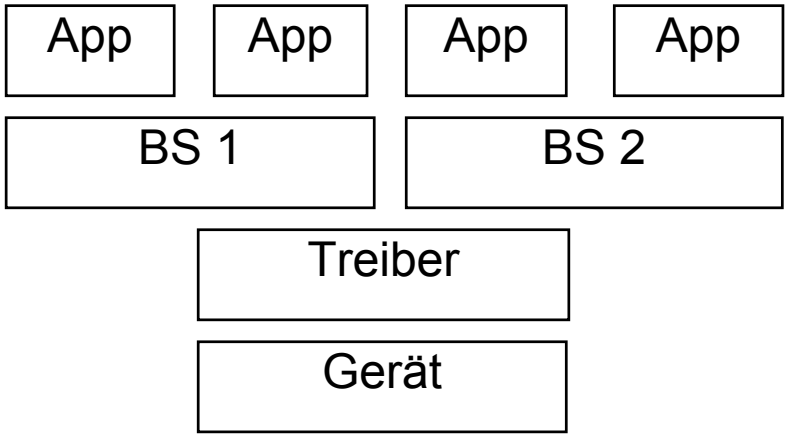

Abbildung 3.10: Zwei Betriebssysteme nutzen denselben Treiber

4.0 Zusammenfassung

Je nach Einsatzgebiet, werden unterschiedliche Emulations- und Virtualisierungsmethoden eingesetzt. Sollen also nur Anwendungsprogramme, keine weiteren Betriebssystemkerne laufen, bietet sich an, nur die API des zweiten Betriebssystems zu emulieren, was im Allgemeinen ausreichend schnell ist. Bei der Emulation wird ein ganzer Rechner komplett in Software nachgebildet Ein Vorteil ist, dass die emulierte Architektur eine ganz andere sein kann, als die, auf der die Emulation läuft. Außerdem werden bei der Emulation gesamte Betriebssysteme mit entsprechenden Anwendungen ausgeführt, was den Emulator zu einem Debug-Werkzeug für Betriebssystemkerne macht. Wenn ein Emulator die gleiche Hardware wie die nachahmt, auf der er läuft, liegt der Gedanke nahe, nicht alles in Software nachzubilden, sondern einen Teil der echten Hardware zu verwenden. Gegenüber der Emulation hat die Virtualisierung den Vorteil, dass eine höhere Geschwindigkeit erreicht wird. Auch die aufwendigen API – Emulationen fallen wie schon bei der Rechner – Emulation weg. Von Nachteil ist jedoch, dass die Performance virtualisierter Rechner gegenüber realer Rechner niedriger ist und die x86-PC-Hardware im Allgemeinen schwer zu virtualisieren ist. Außerdem wurden Funktionen zur Abbildung virtueller auf reale Maschinen vorgestellt und diskutiert. Der Grundsatz lautet, soweit wie möglich die reale Maschine zu nutzen. Was auf diese Weise nicht ausgeführt werden kann oder darf, muss emuliert werden. Kritische und Unkritische Befehle wurden unterschieden. Das Auffinden dieser Befehle ist durch die lineare Suche oder besser durch die Verfolgung des Ausführungspfades möglich. Der Virtual Machine Monitor muss auch sicherstellen, dass nur die von ihm vorgesehenen Seitenrahmen in die aktiven Seitentabellen eingetragen werden. Die grundlegende Idee ist, für diesen Zweck eine dritte Stufe der Adressübersetzung einzuführen.

Auch die Virtualisierung von Geräten war Gegenstand der Arbeit, denn ohne Geräte zur Ein- und Ausgabe bleibt der virtuelle Rechner für den Anwender unbenutzbar. Zur Virtualisierung sind mehrere Architekturen möglich, die in diesem Abschnitt diskutiert wurden.

Literatur

K. Lawton, Running muliple operating systems concurrently on an IA32 PC using virtualization techniques, http://plex86.org/research/paper.txt

Jeremy Sugerman, Ganesh Venkitachalam and Beng-Hong Lim, Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, USENIX Annual Technical Conference, Juni 2001

Andrew S. Tannenbaum, Modern Operating-Systems, New Jersey, Prentice Hall, 2001

VMware Inc., VMware Virtual Machine Technology, http://www.vmware.com

WINE: Windows on Unix, www.winhq.com

Plex86s-Homepage, http://www.plex86.org

Bochs-Homepage, http://www.bochs.org