INSTITUT FÜR INFORMATIK

DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Machbarkeitsanalyse zur Virtualisierung des IT-Sicherheit Praktikums

Tobias Lindinger

Betreuer: Dr. Helmut Reiser
Nils gentschen Felde

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Machbarkeitsanalyse zur Virtualisierung des IT-Sicherheit Praktikums

Tobias Lindinger

Betreuer: Dr. Helmut Reiser
Nils gentschen Felde

Zusammenfassung

Aufgabe dieser Arbeit ist es zu analysieren, ob es möglich ist das Rechnernetz, das im Praktikum IT-Sicherheit verwendet wird, zu virtualisieren. Im positiven Falle soll gezeigt werden, welche Hilfsmittel dazu benötigt werden und wie diese eingesetzt werden können. Ziel sollte es sein, die gesammte Infrastruktur des Netzes auf einem Serversystem nachzubilden und so die veralteten Rechner im Praktikumsbetrieb zu ersetzen. Gleichzeitig sollte - sofern machbar - die mögliche Teilnehmerzahl erhöht werden. Der Zugriff auf die virtuellen Maschinen soll von Arbeitsstationen im CIP-Pool oder von zu Hause erfolgen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Problemstellung und Einführung

Seit einiger Zeit häufen sich im Praktikum IT-Sicherheit Probleme mit der im Praktikum verwendeten Hardware. Insbesondere gehen immer wieder einzelne Komponenten der verwendeten Arbeitsplatzrechner kaputt. Dies hat zur Folge, dass die Betreuer oft mehr damit beschäftigt sind Hardware-Probleme zu lösen, als dass sie ihrer eigentlichen Aufgabe nachkommen können die Studenten bei der Lösung der praktischen Aufgaben zu unterstützen.
Desweiteren ist die Nachfrage bezüglich des Praktikums relativ hoch. Es gibt meist mehr Bewerber als freie Plätze zur Verfügung stehen. Das Problem ließe sich zwar durch die Anschaffung neuer PCs beheben, kostet aber Geld und schafft auch nicht mehr Praktikumsplätze, denn für zusätzliche Arbeitsplätze sind schlichtweg keine Räumlichkeiten mehr vorhanden.
Ein Ansatz zur Lösung beider Probleme wäre eine komplette Virtualisierung des Praktikums; das bedeutet, dass alle im Praktikum benötigten Hardware-Komponenten softwareseitig auf einem Serversystem erzeugt werden und über das Netz zugreifbar sind. Wenn alle einzelnen Komponenten, angefangen vom Netzwerkkabel über Switches, Hubs und Netzwerkkarten sowie Rechner und Server richtig konfiguriert sind, so kann im Zusammenspiel ein komplexes Netz entstehen. Von außen, das heißt über das Netz betrachtet, ist die Virtualisierung transparent; die simulierten Komponenten scheinen real zu existieren. Die einzelnen virtuellen Rechner lassen sich über das Netz bedienen wie jeder physikalisch existierende Rechner auch. So sollte es auch möglich sein die Aufgaben des Praktikums auf solch einem virtuellen System zu bearbeiten.
Auf diese Art könnten vorhandene PCs im CIP-Pool oder zu Hause verwendet werden um auf den virtuellen Systemen des Praktikums zu arbeiten. Man benötigt keine weiteren Rechner und separate Räumlichkeiten mehr für das Praktikum. Lediglich die Serverhardware zur Bereitstellung der virtuellen Netzwerke muss untergebracht werden. Vor der praktischen Umsetzung dieses Vorhabens muss aber eine gründliche Analyse erfolgen, ob und wie dieses Vorhaben umsetzbar ist. Die wichtigsten zu klärenden Punkte sind:

- Welche Virtualisierungs-Tools können die komplette Infrastruktur des Praktikums in Software nachbauen?
- Welche Anforderungen werden durch die Virtualisierung an die Hardware des Serversystems gestellt?
- Welche Änderungen sind eventuell an den Praktikumsaufgaben und am Ablauf notwendig?
- Wie kann man auf den virtuellen Maschinen arbeiten?

Dies heraus zu finden ist Aufgabe dieser Arbeit. Als Testumgebung dienen zwei Pentium 4 PC-Systeme mit 3GHz Taktfrequenz, jeweils einem Gigabyte RAM sowie einer IDE-Festplatte. Als Betriebssystem kommt SuSE Linux 9.3 zum Einsatz, da SuSE schon in weiten Teilen des Instituts eingeführt und damit den meisten Studenten vertraut ist. Zumal bietet SuSE Linux ab der Version 9.2 bereits Unterstützung bei der Konfiguration für einige Virtualisierungs-Tools. Genauer gesagt existieren YAST Module für das Anlegen von User Mode Linux und XEN Umgebungen. Genauere Informationen hierüber liefern die nachfolgenden Kapitel.
Im weiteren Verlauf dieser Arbeit soll zunächst Klarheit über die benötigten Komponenten zum Aufbau der virtuellen Infrastruktur geschaffen werden. Anschließend betrachten wir kurz die verschiedenen Möglichkeiten virtuelle Rechner und Netze zu erstellen und werden anhand der Anforderungen ein Tool für die Umsetzung unseres Vorhabens auswählen. Anschließend werden wir uns mit diesem Tool genauer auseinandersetzen und die technischen Fragen hinsichtlich der Realisierung klären. Am Ende soll dann die Entscheidung fallen, ob und wie eine Virtualisierung der Infrastruktur des Praktikums möglich ist.

Kapitel 2

Grundlagen

2.1 Szenarien und Aufgabenstellungen

Aufgabe des Praktikums IT-Sicherheit ist es, bei den Studenten ein Bewusstsein für mögliche Sicherheitsrisiken in der IT zu entwickeln und einfache Lösungsansätze näherzubringen. So beginnt der erste Teil des Praktikums mit einem Szenario (Abb. 2.1), das einen Hub als Hauptkomponente hat. Ziel ist es zu erkennen, dass jeder am Hub angeschlossene Rechner die Kommunikation der fremden Rechner mithören kann. Dies ist eine wichtige Erkenntnis, auch wenn Hubs mitlerweile von billiger werdenden Switches und DSL-Routern mit integriertem Switch verdrängt werden. Im Falle eines leicht provozierbaren Überlaufens der NAT-Table arbeiten viele von ihnen ebenfalls als Hub und sind somit ein Sicherheitsrisiko. Dem wird im Praktikum damit Rechnung getragen, dass sich die Teilnehmer untereinander belauschen und FTP bzw. telnet Passwörter abhören sollen. Auf diese Art soll gezeigt werden, dass unverschlüsselter Austausch von sensiblen Daten wie zum Beispiel Passwörtern kritisch ist. Nebenbei beschäftigt man sich mit verschiedenen Protokollen sowie Aufbau von IP basierten Netzen. Für die Virtualisierung des Praktikums ist es daher unbedingt nötig, dass das Virtualisierungstool Hubs erzeugen kann.

Im zweiten Szenario (Abb. 2.2) des Praktikums arbeiten die Teilnehmer mit einem Switch als Hauptkomponente des Netzes. Hauptaugenmerk liegt hier auf Routing sowie dem Erstellen von statischen und dynamischen Firewalls mit Hilfe von `iptables`. Desweiteren wird auf die Konfiguration von einigen wichtigen Diensten wie Web- und Mail-Server eingegangen. Weitere Themen sind VPN/Verschlüsselung, DNS, Intrusion Detection, Proxies und Application Level Gateways. Für eine Virtualisierung dieses Szenarios ist es deshalb wichtig, dass das Virtualisierungstool Switches erzeugen kann. Desweiteren muss gängige Linuxsoftware auf den virtuellen Maschinen lauffähig sein.

In beiden Szenarien existiert ein Server namens „secserver", der für bestimmte Aufgaben einfache Dienste bereitstellt; beispielsweise HTTP-Proxy, DNS, NFS usw. Auch er muss in das virtuelle Netz des Praktikums eingebunden werden. Hierfür bieten sich zwei unterschiedliche Möglichkeiten: Variante eins virtualisiert den gesamten Server mit seinen Diensten. Variante zwei stellt lediglich seine Dienste auf dem Virtualisierungsserver zur Verfügung. Beide Varianten haben Vor- und Nachteile, eine begründete Auswahl können wir jedoch erst treffen, wenn wir uns auf ein Virtualisierungstool festgelegt und erste Erfahrungen damit gemacht haben.

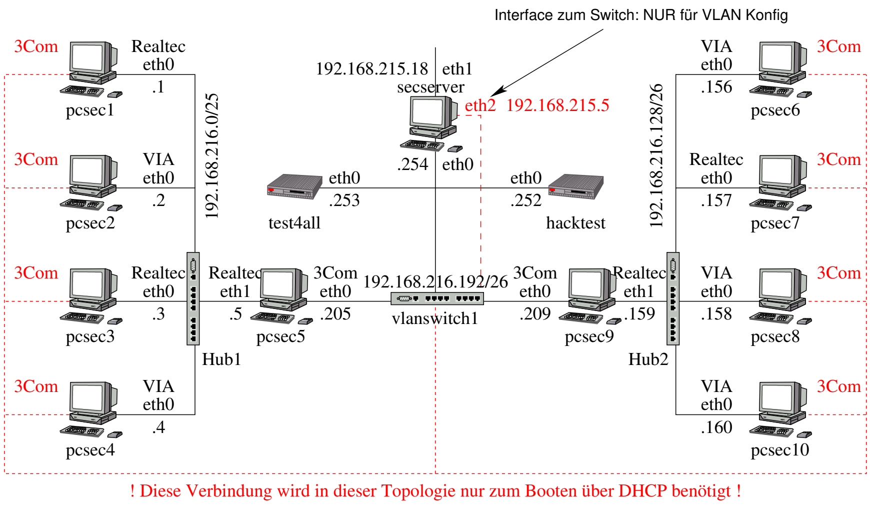


Abbildung 2.1: Szenario 1

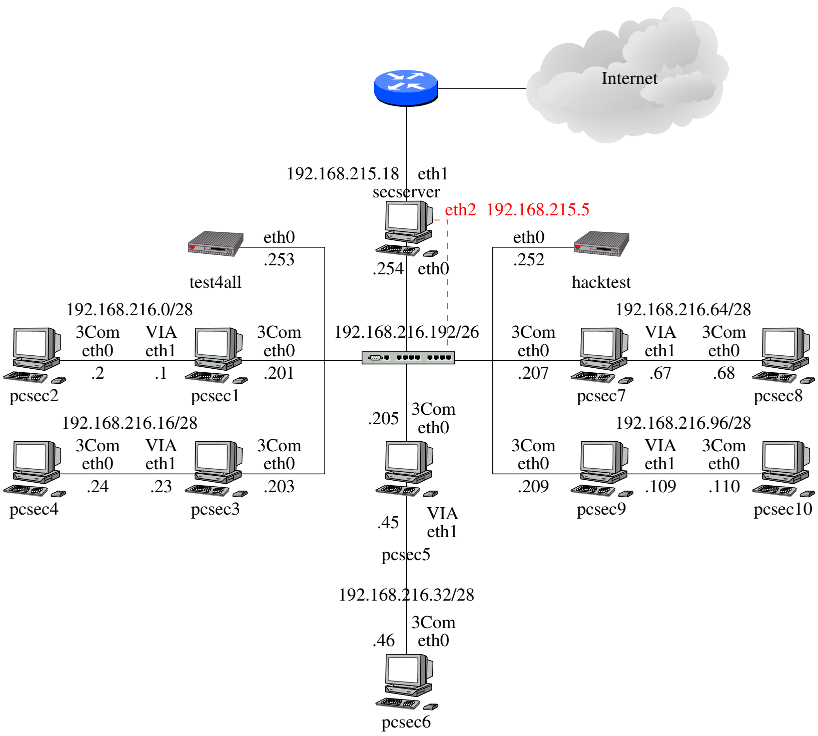


Abbildung 2.2: Szenario 2

Bisher arbeiteten jeweils zwei Studenten an einem Rechner. Dies hat zur Folge, dass sie gerade im zweiten Szenario bei einigen Aufgaben nur Erfahrung mit der Serverkonfiguration einer Software hatten oder nur Clienterfahrung. Um dies in Zukunft zu ändern, soll das Praktikum so umorganisiert werden, dass zwei Studenten jeweils einen Client und einen Server konfigurieren können. Ein Team von zwei Studenten bekommt dazu ein Teilszenario wie etwa „pcsec1” und „pcsec2” zugewiesen. Das gesammte Szenario vergrößert sich damit, da weitere Client-Server Teams an den Switch angeschlossen werden müssen. Die Virtualisierung muss in Summe also 40 Arbeitsrechner eventuell den „secserver”, sowie die Rechner „test4all” und „hacktest” erzeugen.

2.2 Virtualisierungstools

Virtualisierungs-Tools gibt es viele, angefangen bei den bekannten kommerziellen Lösungen VMware [vmw05] und MS Virtual PC [vir05] über das freie Bochs [boc05] sowie User Mode Linux [uml05] und XEN [xen05], um die wichtigsten zu nennen.
Sie alle können über das Netz gesehen Rechner simulieren, bedienen sich dabei aber verschiedener Techniken.
VMware, Virtual PC und Bochs versuchen die komplette i386-Architektur mit allem Zubehör in Software nachzubilden. Anschließend kann man jedes für diese Plattform geeignete Betriebssystem seiner Wahl als Gastbetriebssystem installieren. Dieses Vorgehen bringt den großen Vorteil mit sich, dass das Gastbetriebssystem nicht modifiziert werden muss und die Virtualisierung im Normalfall transparent ist. Allerdings sind alle drei Tools nicht für unser Vorhaben viele virtuelle Maschinen gleichzeitig laufen zu lassen geeignet, da sie sehr hohe Ansprüche an Arbeitsspeicher und Prozessor stellen.
Anders sieht es mit User Mode Linux und XEN aus. Während User Mode Linux auf einen im User Mode laufenden leicht angepassten Linux-Kernel, der auf jedem normalen Linuxsystem laufen sollte, setzt, verwendet XEN ein leicht modifiziertes Linuxsystem, das die der i386-Architektur sehr ähnliche XEN Plattform, auf der dann später die viruellen Maschinen aufsetzen, bereitstellt. Beide Varianten sind insofern für das Projekt geeignet, als dass sie auch mit einer größeren Anzahl virtueller Maschinen zurechtkommen. XEN disqualifiziert sich jedoch aus einem anderen Grund:
Es existieren zwar Switches und Bridges für den Einsatz in XEN, bisher aber noch keine Hubs, wie wir sie in unserem ersten Szenario (Abb. 2.1) benötigen.
Der Rest dieser Arbeit wird sich daher nur noch damit beschäftigen, ob die Virtualisierung des Praktikums mit User Mode Linux machbar ist.

Kapitel 3

User Mode Linux

User Mode Linux [uml05], im Folgenden auch kurz UML genannt, wird von einer kleinen Gruppe von Entwicklern um Jeff Dike entwickelt. Das Projekt ist, da es direkt mit den Linux-Kernel-Sourcen arbeitet, frei unter der GPL[1] erhältlich und wird auf Sourceforge gehostet.
Ziel der Entwickler ist es nicht in erster Linie eine virtuelle Maschine zu erstellen; sie wollen User Mode Linux eher als virtuelles Betriebssystem verstanden wissen. Ihr Ziel ist es eine Art Sandbox für bestimmte Anwendungen, zum Beispiel Webserver, zu entwickeln. So könnte ein Provider mehrere von einander getrennte Webserver auf einem einzigen Server laufen lassen und seinen Kunden jeweils die Administrationsrechte für einen virtuellen Server gewähren. User Mode Linux eignet sich daher hervorragend für unseren Zweck, viele Rechner zu virtualisieren, sofern sie keine allzu rechenintensiven Aufgaben zu bewältigen haben, da es selbst relativ wenig Rechenleistung zur Verwaltung der virtuellen Maschinen benötigt. Abb. 3.1 zeigt ein laufendes User Mode Linux System. Unten im Bild sieht man die aufrufende Konsole, links oben die Bootmeldungen der UML und rechts oben ein `xterm`, das den Zugriff auf ein Terminal der virtuellen Maschine bietet.

3.1 Architektur

Im Folgenden soll kurz auf die zu Grunde liegende Architektur von User Mode Linux eingegangen werden. Das Verständnis über die internen Abläufe von User Mode Linux ist unbedingt von Nöten um die virtuellen Maschinen sowie den Host möglichst effizient konfigurieren und damit möglichst viele virtuelle Systeme auf einem Server laufen lassen zu können.

3.1.1 Allgemeines

Linux läuft bereits heute auf vielen unterschiedlichen Plattformen. Die Anpassung an eine neue Plattform ist vergleichsweise einfach zu implementieren, wenn man den gesamten Umfang des aktuellen Kernels betrachtet. Es muss lediglich die Architekturschicht neu implementiert werden.
Genau das macht User Mode Linux. Es implementiert die arch-Schnittstelle des Linux-Kernels und ist daher in der Lage auf einer virtuellen Plattform zu laufen. Dabei werden nicht einzelne Hardwarekomponenten in Software nachgebaut und zu einem lauffähigen Gesamtsystem zusammengesetzt,

[1] General Public License

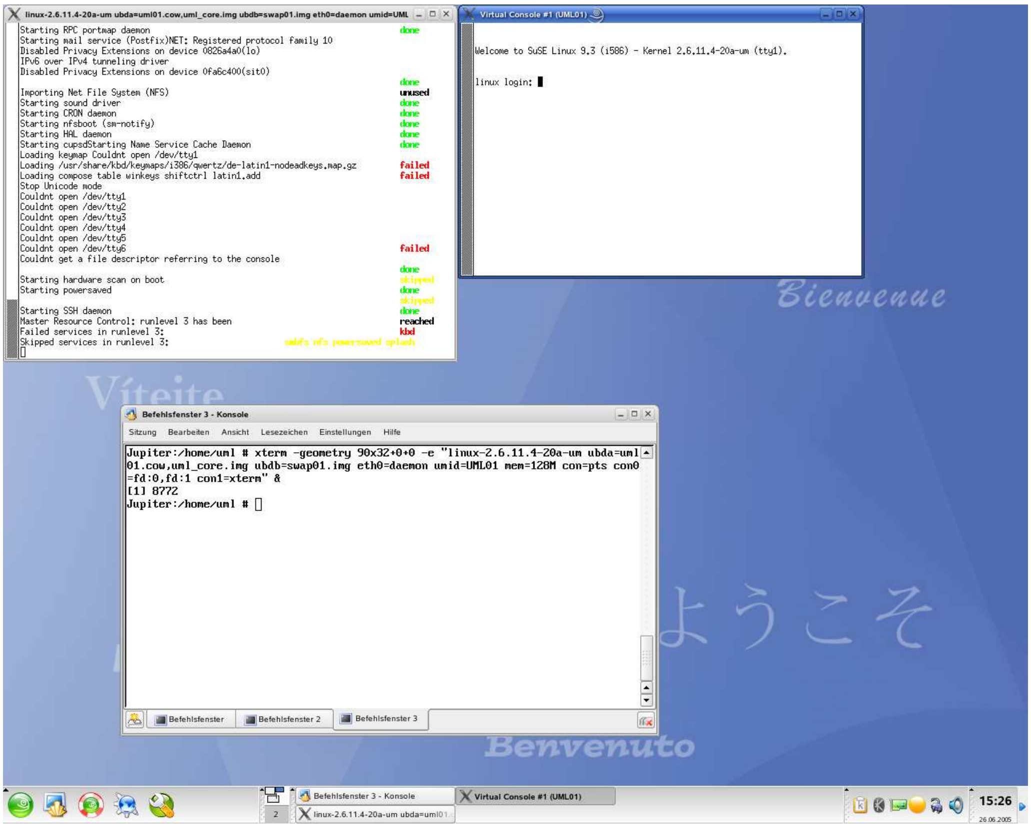


Abbildung 3.1: Virtuelles Linux System

wie dies etwa VMWare löst, sondern der User Mode Linux Kernel läuft als normales Programm im User Mode. Durch die Architekturschicht werden Systemaufrufe wie etwa Festplattenzugriffe abgefangen und entsprechend behandelt. Oberhalb der arch-Schnittstelle ist die Virtualisierung nicht mehr bemerkbar, d.h. es laufen wie in jedem „normalen" Linux ein Scheduler, Rechteverwaltung und diverse andere Prozesse. Einen groben Überblick über die Architektur gibt Abb. 3.2. Die Abbildung zeigt deutlich, dass der User Mode Linux Kernel auf der selben Hirachiestufe läuft wie normale Anwendungsprogramme. Als Beispiel sind hier `ls`, `ps` und `mozilla` aufgeführt. Erst oberhalb der eigentlichen Programmebene laufen die Prozesse des virtuellen Linux-Systems. Sie sind eigentlich für das zugrunde liegende Host-Linux nicht sichtbar. Eigentlich aber nur deshalb, weil zwar die Architektur der verschiedenen Hirachieschichten verhindern sollte, dass diese Programme unterhalb der User Mode Linux Architektur sichtbar sind, aber User Mode Linux für jeden virtuell gestarteten Prozess zur Ausführung einen Klon im Host-Linux erzeugt. Insofern sind die virtuellen Prozesse doch nicht komplett abgesondert. Daraus resultieren auch einige Performance- und Sicherheitsprobleme, die sich aber mittels des SKAS-Patches beheben lassen. Seine Wirkung wird in Abschnitt 3.1.3 behandelt.

Um ein virtuelles Linuxsystem zu erzeugen, benötigt man zwingend zwei Komponenten: Ein virtuelles Dateisystem - es wird meist in Form einer Imagedatei bereitgestellt - und den User Mode Linux Kernel sammt passenden Modulen. Der Start eines UML-Systems erfolgt grundsätzlich sehr ähnlich dem eines normalen Linux-Systems. Der User Mode Linux Kernel versteht aber im Gegensatz zum

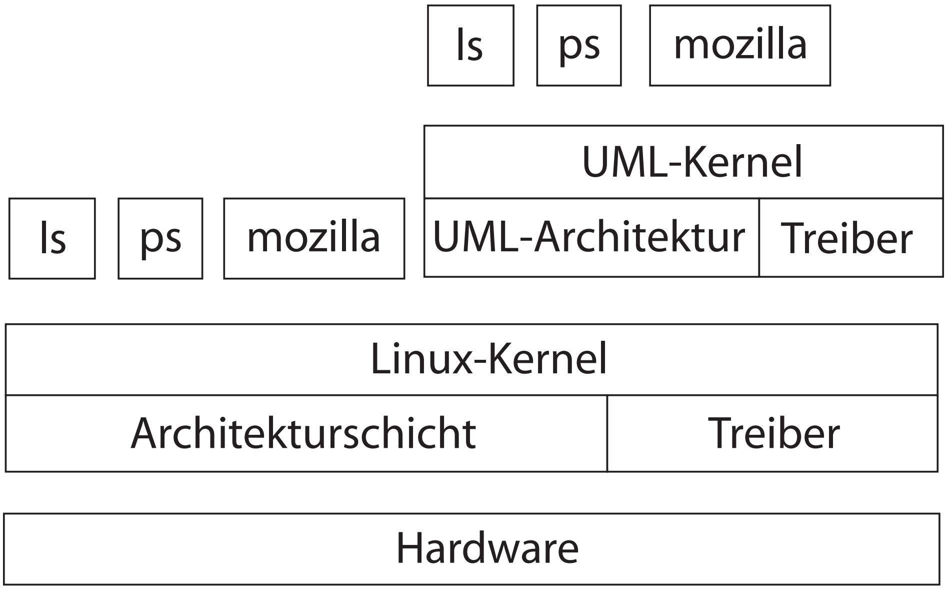


Abbildung 3.2: User Mode Linux Architektur

normalen Kernel jedoch einige zusätzliche Argumente, die ihm mitteilen welche Hardware zu simulieren ist. Genaueres hierzu liefert Abschnitt 3.5.

3.1.2 Systemaufrufe

Wie in vielen Programmen treten auch in Programmen innerhalb der virtuellen Maschine Systemaufrufe auf. Diese müssen entsprechend behandelt werden, da ein Systemaufruf, der von einem Prozess innerhalb der virtuellen Linux-Maschine initiiert wird, nicht wie gewollt in den Kernel-Mode des virtuellen System schaltet, sondern in den Kernel-Mode des darunterliegenden Hostsystems. Der Systemaufruf würde also im Kontext des falschen Betriebssystems ausgeführt werden.
Das Problem löst man mit Hilfe eines Tracing-Threads, der nichts anderes zu tun hat, als auf Systemaufrufe der UML zu warten. Der Tracing-Thread wird nach einem Systemaufruf vom Kernel des Hosts über die eigentlich zu Debugging-Zwecken geschaffene ptrace-Schnittstelle informiert, sobald in den Kernel Mode geschalten werden soll. Darauf hin neutralisiert der Tracing-Thread den Systemaufruf mit Hilfe eines getpid-Aufrufes und behandelt den Systemaufruf im Kontext des UML-Kernel. Anschließend gibt er die Kontrolle über den Programmfluss zurück an die UML, wo der aufrufende Prozess weiter abgearbeitet wird. Der gesamte Ablauf wird schematisch in Abb. 3.3 dargestellt.
Beim Systemaufruf innerhalb eines User Mode Linux Prozesses geht die Kontrolle an den im Hostsystem laufenden Tracing-Thread, der den Systemaufruf im Host neutralisiert. Stattdessen übergibt er den Aufruf an den Kernel des virtuellen Systems, also User Mode Linux. Nach der Abarbeitung gelangt die Kontrolle des Programmflusses entgegengesetzt der ursprünglichen Aufrufreihenfolge über die Stacks zuerst wieder zum Tracing-Thread und anschließend in den aufrufenden Prozess, der weiter abgearbeitet wird.

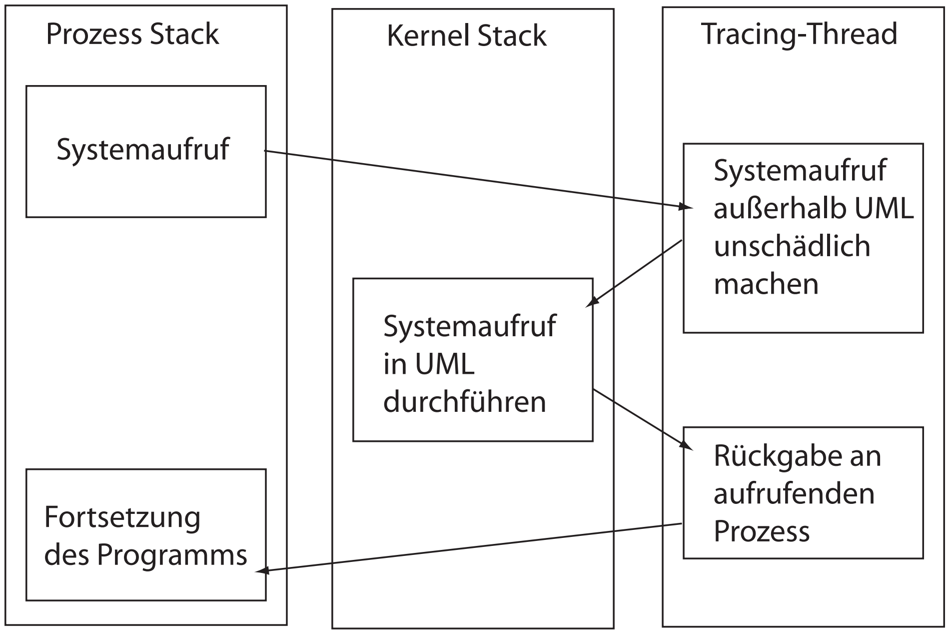
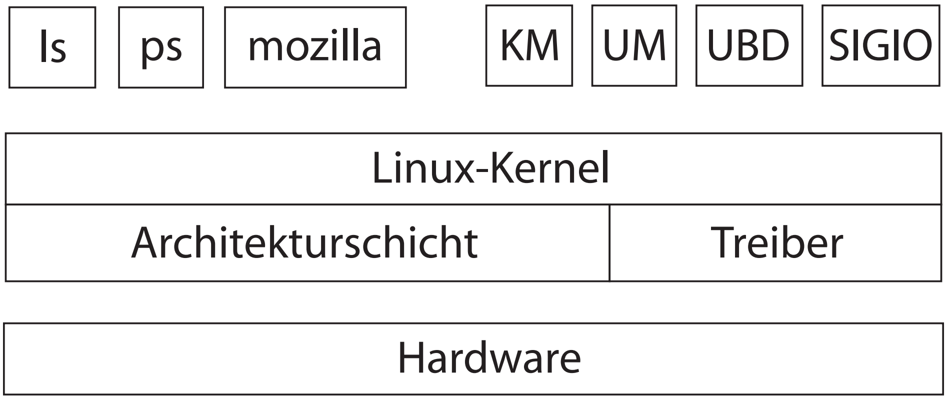


Abbildung 3.3: Systemaufrufe in User Mode Linux

3.1.3 Der SKAS-Patch

Durch die bisher vorgestellte Architektur von User Mode Linux entsteht jedoch ein gravierendes Sicherheits-Problem. Es existieren keine von einander separierten Adressräume. So kann Prozess A ohne Einschränkung auf Daten von Prozess B zugreifen und diese sogar manipulieren; nicht einmal der Speicherbereich des Kernels ist vor solchen Angriffen geschützt. Dies zu lösen ist Ziel des SKAS[2]-Patch [ska05].
Mit ihm wird die Möglichkeit geschaffen mehrere von einander getrennte Adressräume für einen einzigen Prozess zu erzeugen. Des weiteren erweitert er die ptrace-Schnittstelle um einige zusätzliche Kommandos, so dass es Prozessen nun möglich ist, sich selbst zu unterbrechen. Durch diese beiden Design-Änderungen verändert sich die Architektur des Systems nachhaltig: Es existiert nur noch ein einziger Prozess zur Ausführung von virtuellen Prozessen innerhalb der UML. Dieser Prozess kann sich nun selbst zu Scheduling-Zwecken per ptrace-Kommando unterbrechen, seinen aktiven Adressraum ändern und dadurch einen anderen UML-Prozess zur Ausführung bringen. Ähnlich verläuft auch ein Wechsel in den Kernel Mode von User Mode Linux. Als Resultat sind auch die im tt-Mode - so nennt sich der ungepatchte Modus von UML - erzeugten doppelten Prozesse nicht mehr notwendig. Für den Host sichtbar sind pro UML insgesamt nur noch vier Prozesse, wie dies in Abb. 3.4 erkennbar ist.
Ein positiver Seiteneffekt des SKAS-Patches ist die deutlich bessere Performance der virtuellen Maschinen. Durch die Möglichkeit der Prozesse sich selbst zu unterbrechen, benötigt man keinen Tracing-Thread mehr und kann sich aus diesem Grund zwei der vier Prozesswechsel bei einem Systemaufruf sparen. Statt bei einem Systemaufruf wie in Abb. 3.3 an den Tracing-Thread zu geben, erhält beim Einsatz des SKAS-Patches unmittelbar der User Mode Linux Kernel die Kontrolle.
Auf der Projekthomepage von User Mode Linux ist von einer Verdopplung der Performance bei der

[2]Seperate Kernel Address Space



KM : Prozess zur Ausführung von Kernel-Code
UM : Prozess zur Ausführung von User-Code
UBD : Treiber-Prozess für Blockgeräte
SIGIO : Prozess zur Signal-Verarbeitung

Abbildung 3.4: User Mode Linux Architektur mit SKAS-Patch

Kompilierung der Kernelquellen bei Verwendung des SKAS-Patches die Rede. In einigen Ausnahmefällen soll sogar eine Vervierfachung der Geschwindigkeit gemessen worden sein.
Zur Zeit liegt der SKAS-Patch in der Version 3 vor. Um ihn nutzen zu können muss er manuell in die Kernelsourcen des Hostsystems eingebunden und kompiliert werden. Eine feste Integration des Patches in den Kernel ist erst für die überarbeitete Version 4 vorgesehen. Eine manuelle Integration bedeutet derzeit zwar noch viel Arbeit, da der komplette Kernel neu übersetzt werden muss, bringt aber für das System Sicherheit und erhöht die Performance. Negative Eigenschaften des SKAS-Patches wurden bisher nicht beobachtet.

3.2 Erzeugen von Dateisystemen für UML

Laut SuSE-Werbung für die Distribution 9.3, ist das Aufsetzen eines UML-Systems mit dem integrierten Assistenten recht einfach. Im Konfigurationscenter YAST auf „UML-Installation" klicken, Netzwerk-Installationsquelle und Größe des Images angeben und schon sollte die Installation der virtuellen Maschine starten. Nach dem Anlegen des Images und initialem Start des virtuellen Systems tritt aber bereits der erste Fehler auf: YAST findet den angegeben NFS-Server nicht und auch die Netzverbindung des Host funktioniert nicht mehr. Der Fehler: YAST legt nicht nur ein virtuelles Netzwerk-Interface an, sondern auch gleich noch eine Bridge, um - falls vorhanden - Kontakt mit einem DHCP-Server aufnehmen zu können. Leider ist die Bridge jedoch alles andere als funktionstüchtig konfiguriert und für unseren Einsatzzweck zudem absolut nutzlos. Das Hindernis lässt sich durch einen manuellen Start der Maschine ohne Bridge umgehen. Anschließend startet die gewohnte Installation in deren Verlauf man die erzeugte Image-Datei partitionieren und formatieren muss. Leider kann man anschließend von diesem Image nicht booten, da der UML ubd-Treiber[3] nur mit

[3] User Block Device-Treiber; regelt den Zugriff auf die virtuelle Festplatte des UML-Systems

unpartitionierten Dateien umgehen kann. Der SuSE-Installations-Assistent ist also derzeit noch keine brauchbare Hilfe zur Installation einer virtuellen Maschine.
Eine Alternative wäre das Installationsskript von Kraxel [Kno03], das ohne nennenswerte Vorkenntnisse eine funktionsfähige UML-Instanz erzeugt. Leider kann man die Installation nicht genügend exakt beeinflussen; so ist es z.B. nicht möglich einzelne Software-Pakete zur Installation hinzuzufügen bzw. entfernen; zur Auswahl stehen lediglich „Minimales System" sowie „Minimales System + X11". Somit ist auch dieses Tool unbrauchbar.
Will man eine wirklich funktionierende und für die Praxis taugliche Installation, so muss man die dazu erforderlichen Schritte selbst in der Kommandozeile erledigen. Zuerst erzeugt man sich mit

```
dd if=/dev/zero of=imagename bs=1M count=4096
```

ein 4GB großes Imagefile. Alternativ kann man auch ein Sparse-Image anlegen, welches erst dynamisch mit seinem Inhalt bis zu einer maximalen Grenze wächst:

```
dd if=/dev/zero of=imagename bs=1M count=0 seek=4096
```

Der Vorteil dieser Variante sehr viel Speicherplatz zu sparen trügt jedoch: Sparse-Images neigen dazu sehr schnell zu fragmentieren. Den gesparten Plattenplatz erkauft man sich also mit längeren Zugriffszeiten und niedrigerem Durchsatz. Ein anschließendes

```
mkfs -t ext3 imagename
```

formatiert das Image. Mit

```
mount -o loop imagename /mnt/
```

kann man es schließlich in das Dateisystem einhängen. Nun steht der Installation der Pakete mit Hilfe des YAST-Moduls „Installation in ein Verzeichnis" (Abb. 3.5) nichts mehr im Wege. Lediglich die Paketauswahl muss per Hand etwas für den Einsatzzweck angepasst werden. So darf man auf keinen Fall den Bootloader GRUB installieren, da dessen Aufgabe schon der externe Part des UML-Kernel übernimmt. Diesen muss man jedoch separat noch im Host-System installieren. Den Standard-Kernel sollte man bei dieser Gelegenheit aus der Liste entfernen, er wird nicht benötigt.
Die Meldungen vom YAST, dass diese Pakete für den Betrieb eines normalen Systems essentiell sind, muss man in diesem Falle ignorieren.
Die Abfrage, ob SuSEconfig beim ersten Systemstart ausgeführt werden soll, bestätigt man mit einem Häckchen. Nachdem alle Pakete installiert worden sind, müssen noch einige kleine Korrekturen am Dateisystem vorgenommen werden, ohne die das UML-System nicht läuft. Da wäre zum einen das Anlegen einer gültigen `/etc/fstab` in der die Zeilen

```
/dev/ubda/  /          ext3    defaults  1 1
proc        /proc      proc    defaults  0 0
devpts      /dev/pts   devpts  defaults  0 0
```

stehen müssen. Ohne den ersten Eintrag bootet die virtuelle Maschine nicht, da sie ihr Root-Filesystem nicht findet und der Bootvorgang mit „kernel panic" abbricht. Die anderen beiden Einträge sind für

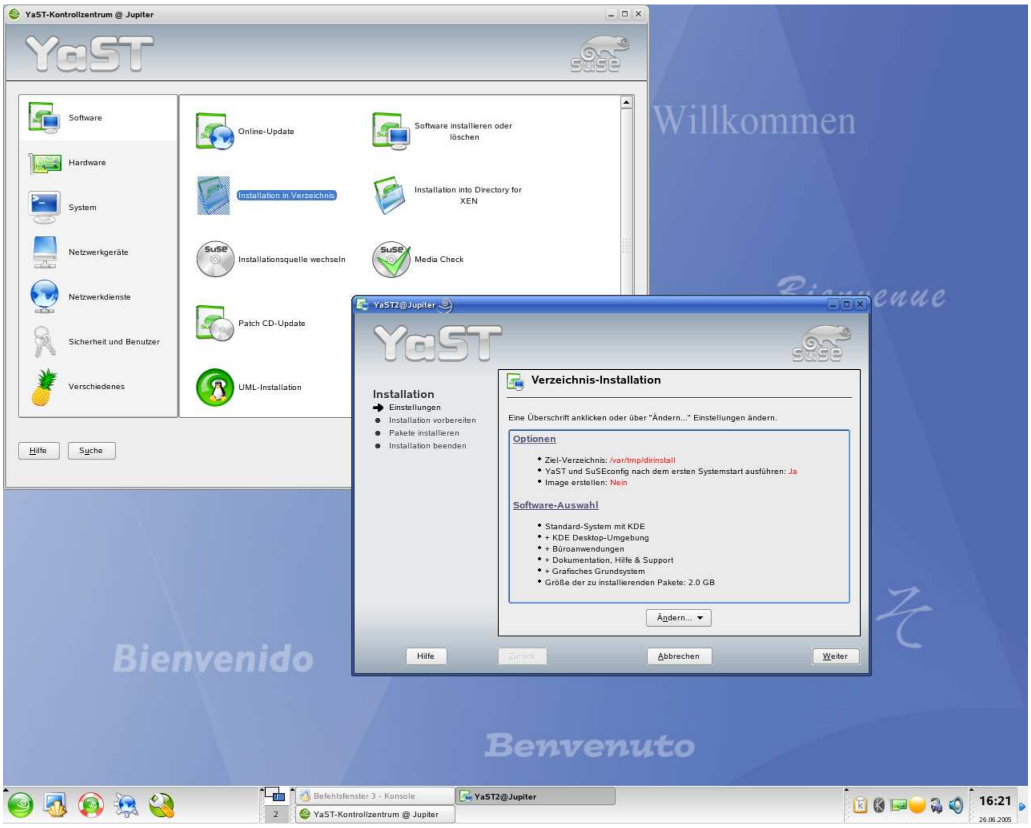


Abbildung 3.5: YAST - Installation in Verzeichnis

interne Verwaltungsvorgänge des Kernels notwendig.
Will man UML eine zusätzliche SWAP-Partition zur Verfügung stellen, so fügt man noch folgenden Eintrag hinzu:

```
/dev/ubdb/  swap  devpts  defaults  0 0
```

Die Swap-Datei selbst wird wie das Image mit

```
dd if=/dev/zero of=swap bs=1M count=128
```

angelegt. Formatiert wird es mit dem Kommando `mkswap`. Zuletzt benennt man das Verzeichnis `/lib/tls` in `/lib/tls.disabled` um. Ohne diesen Schritt, der das TLS[4] deaktiviert und intern für ein Fall Back auf ein älteres Verfahren des Thread-Managements sorgt, werden Anwendungen die aus mehreren Threads bestehen nicht richtig ausgeführt und liefern nur ein Segmentation fault. Beispiele für solche Anwendungen sind haupsächlich Programme, die zum Betrieb von Netzwerken erforderlich sind, wie z.B. `host`, aber auch YAST erzeugt ohne diese Änderung beim Startversuch nur Fehlermeldungen. Mit diesen Schritten ist das Image fertig für den ersten Start. Zuvor muss es jedoch unbedingt mit

[4]Thread Local Storage

```
umount /mnt/
```

aus dem Dateisystem des Hosts ausgehängt werden, so dass die UML das exklusive Schreibrecht auf die Datei bekommen kann. Aus dem selben Grund kann ein Image nicht gleichzeitig für mehrere UML Instanzen verwendet werden, denn mehrere schreibende Instanzen auf einem Dateisystem würden Inkonsistenzen erzeugen. Um dem entgegen zu wirken wurde hierfür eigens das Konzept der COW[5]-Files entwickelt. Alle UML-Instanzen verwenden dasselbe Image („Core-Image"), auf das sie jedoch nur lesend Zugriff haben. Alle Änderungen, die vorgenommen werden sollen, werden für jedes System separat in eine Datei geschrieben, die lediglich das Delta zum eigentlichen Image speichert. Diese Variante ist sehr viel effizienter, als für jedes laufende System ein eigenes Image anzulegen, das jeweils nicht nur Platz auf der Festplatte beansprucht, obwohl es bis auf wenige Dateien identisch mit allen anderen Images ist, sonderen auch noch identische Dateien mehrfach im Hauptspeicher gehalten werden. Die COW-Files belegen zwar scheinbar jeweils den gleichen Plattenplatz wie das Core-Image selbst, sind aber in Wirklichkeit sehr viel kleiner und wachsen erst mit der Anzahl der Änderungen bis auf ihre zugewiesene Maximalgröße.

Achtung: Ändert man nach der Erstellung von COW-Files etwas am Image, so werden alle COW-Files unbrauchbar! Es könnten Änderungen an Stellen im Dateisystem des Core-Images auftreten, die schon durch ein COW-File geändert werden. Herauszufinden, welche Änderungen davon nun Gültigkeit haben sollen, ist nicht möglich. Aus diesem Grund wird das nachträgliche Ändern von Core-Images nicht unterstützt (siehe auch Abschnitt 3.5).
Nicht vorhandene COW-Files werden beim ersten Systemstart automatisch erzeugt.
Eine Integration des Deltas in das Image ist mit dem Tool uml_foo möglich. Es erzeugt aus einem Core-Image und einem COW-File wahlweise ein neues Image, dass die Änderungen des COW-Files bereits beinhaltet, oder integriert die Änderungen direkt in das vorhandene Core-Image. Auch hier gilt jedoch: Alle anderen Delta-Dateien, die vom selben Core-Image abhängen, werden ungültig. Mit Hilfe von uml_foo lassen sich Änderungen erst mit COW-Files testen, bevor man sie fest in das Image übernimmt. Das spart das ständige Sichern ganzer Core-Images beim Testen von komplexen Konfigurationen.

3.3 Erstellen virtueller Netzwerke

Möglichkeiten UML-Systeme an ein Netz anzubinden gibt es viele. Für unseren Fall eignen sich besonders die Varianten TunTap sowie uml_switch. TunTap arbeitet als Kernelmodul und ist im SuSE-Kernel bereits integriert; uml_switch muss man erst auf dem Host installieren, was YAST aber automatisch erledigt, wenn man bei der Installation den Punkt UML-Unterstützung auswählt. TunTap stellt ein virtuelles Netzwerk-Interface bereit, während uml_switch einen Switch oder wahlweise mit der option -hub einen Hub simuliert. Soll der Host mit in das virtuelle Netz integriert werden, so erstellt man zunächst mit

```
tunctl -t uml0
```

eine virtuelle Netzwerkkarte mit dem Namen uml0. Dieser weist man mittels

[5]Copy on Write

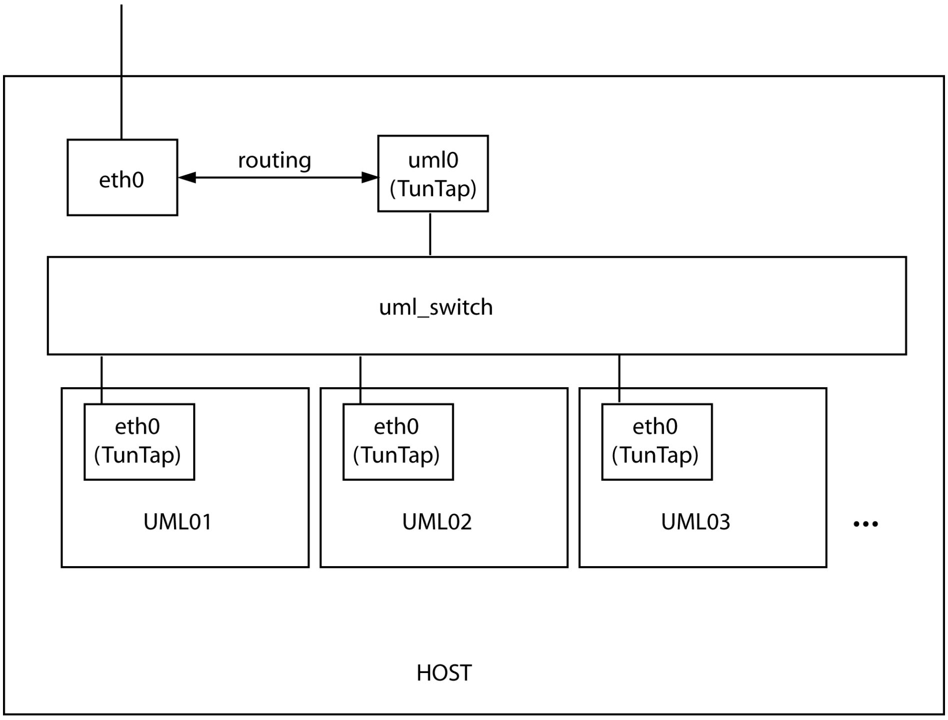


Abbildung 3.6: Virtuelles Netz mit uml_switch

`ifconfig uml0` *`IP-Adresse`* `up`

eine Adresse zu und schaltet das Interface aktiv.
Anschließend startet man den Switch mit

`uml_switch -tap uml0.`

Kurz darauf sieht man wie die Schnittstelle uml0 mit einer künstlichen MAC-Adresse mit dem Switch verbunden wird.

Hinweis: Es können mehrere virtuelle Switches angelegt werden, allerdings muss dann explizit ein Socketname übergeben werden. Das Kommando zum Starten des Switches lautet dann

`uml_switch -tap uml0 -unix` *`socketname`*,

wobei *`socketname`* eine beliebige Datei sein kann. Default ist `/tmp/uml.ctl`.

Abb. 3.6 zeigt den prinzipiellen Aufbau eines virtuellen Netzes mit Hilfe von `uml_switch`. Für die Kommunikation der virtuellen Maschinen untereinander wird das Interface uml0 des Hosts nicht benötigt. Alternativ kann auch ein TunTap-Interface einer virtuellen Maschine direkt mit einem TunTap-Interface des Hosts verbunden werden. Dies entspricht dem „Cross-Over"-Kabel in der Realität. Allerdings benötigt man um die selbe Anzahl Maschinen zu vernetzen auf diese Art deutlich mehr virtuelle Interfaces im Host, die natürlich auch IP-Addressen benötigen. Zudem muss man jetzt auch noch das Routing selbst konfigurieren. Trotzdem hat diese Realisierung ihre Daseinsberechtigung für den Aufbau eines Managementnetzes, da man nur so verhindern kann, dass die einzelnen UMLs über das Managementnetz miteinander kommunizieren.

3.4 Grafische Oberflächen

Auf UML-Systemen kann generell kein X11-Server gestartet werden, da sie nicht über eine eigene Grafikkarte verfügen. Auf grafische Oberflächen muss man dennoch nicht vollends verzichten. So ist es zum Beispiel möglich sich via `ssh -X` auf den Maschinen anzumelden. Ein laufendes X11-System auf dem Remote-Client vorausgesetzt, kann man so alle Fenster, die vom virtuellen System geöffnet werden direkt auf den Client exportieren. Das Vorgehen unterscheidet sich nicht von dem mit realen Rechnern.
Benötigt man jedoch einen richtigen, virtuellen Windowmanager, so ist zusätzlicher Konfigurationsaufwand nötig: Zuerst installiert man auf dem Client das Paket X11-Xnest[6] und startet auf dem Remote-Client unter laufendem X11 einen virtuellen X-Server mittels

```
Xnest -ac :1.
```

In der UML-Maschine setzt man als Displayvariable das virtuelle Xnest-Display des Clients mit

```
export DISPLAY=IP-Adresse:1.
```

Anschließend kann man einen beliebigen Windowmanager starten, für KDE z.B. mit

```
startkde.
```

Das Ergebnis zeigt Abb. 3.7.
Diese Variante hat jedoch zwei bisher nicht gelöste Probleme: auf dem virtuellen KDE kann man ausschließlich mit englischem Tastaturlayout arbeiten und bisweilen muss man neu erstellte Fenster minimieren und maximieren um deren Inhalt sehen zu können.
Ein weiterer Nachteil besteht darin, dass die gängigen Windowmanager sehr viel Arbeitsspeicher benötigen. Für unser Vorhaben kommt daher eigentlich nur die erste Option in Frage.

[6] Xnest ist eine Kombination aus X-Server und X-Client. Auf Clientseite benutzt es den lokalen X-Server das Systems um sein Fenster darzustellen; für UML arbeitet Xnest als X-Server, auf dem die virtuelle Maschine ihre Fenster darstellen oder auch einen seperaten Windowmanager starten kann.

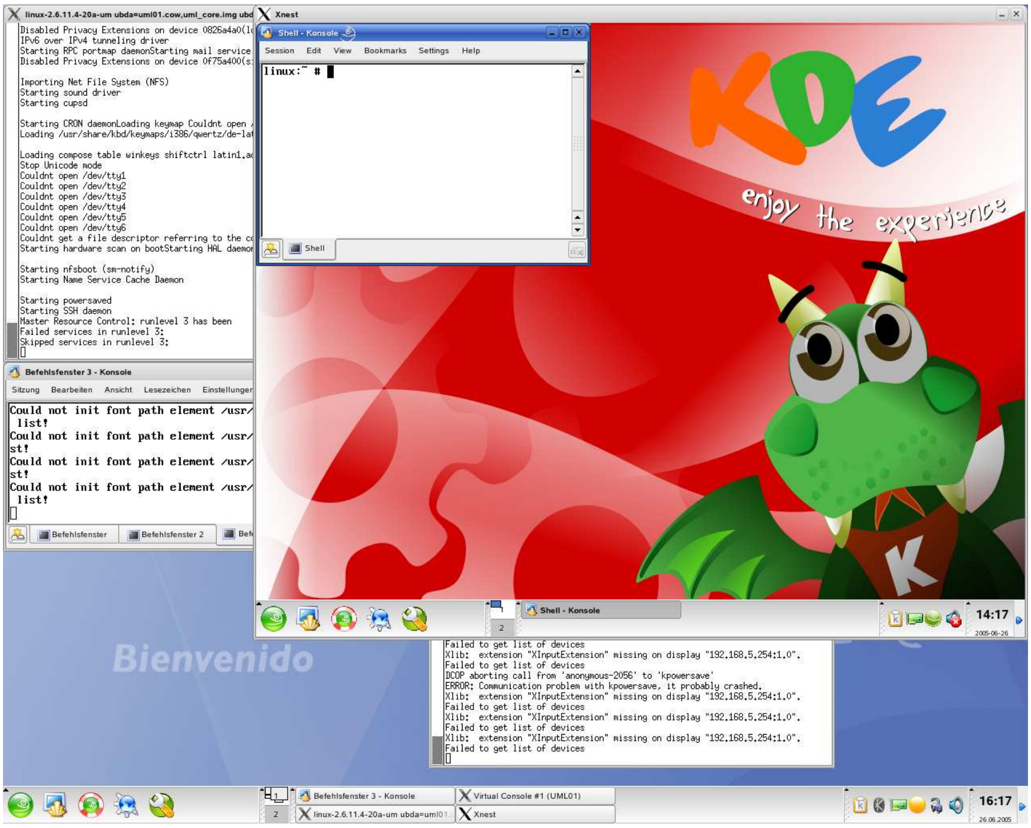


Abbildung 3.7: User Mode Linux mit virtuellem Windowmanager KDE

3.5 Starten einer virtuellen Maschine

Der Start einer virtuellen Maschine erfolgt, abgesehen von der Versionsnummer des Kernels, durch das Kommando

`linux-2.6.11.4-20a-um`.

Als Parameter kann man fast alle gängigen Kernelparameter übergeben. Zusätzlich versteht die UML-Variante des Kernels einige zusätzliche Optionen, von denen die Option zur Angabe des Filesystem-Images zwingend ist. Nachfolgend werden die wichtigsten erklärt.

- `ubda=`*`umlcow`*`,`*`umlcore`*

  übergibt als erste Partition einer IDE-Platte das Image *`umlcore`* mit den Änderungen des COW-Files *`umlcow`*. Soll nur ein normales Image, ohne Verwendung von COW-Files übergeben werden, entfällt der erste Parameter samt Komma.

- `ubdb=`*`swap`*

  übergibt ein einfaches Image, das wie hier als Swap-Partition genutzt werden kann, sofern die Einträge in `/etc/fstab` wie bereits in Abschnitt 3.2 beschrieben angelegt wurden.

- `eth0=daemon,`*`MAC-Adresse`*`,`*`Sockettyp`*`,`*`Socketname`*

  legt eine virtuelle Netzwerkkarte mit Namen `eth0` an und verbindet sie mit einem Switch.
  Die letzten drei Parameter sind optional. Unterlässt man die Angabe einer MAC-Adresse, so wird automatisch eine generiert. Als Wert für Sockettyp wird derzeit nur *`unix`* unertsützt. *`Socketname`* identifiziert den gewünschten Switch, default ist `/tmp/uml.ctl`.

- `umid=`*`UML01`* gibt der virtuellen Maschine den Namen UML01, über den sie durch die Management-Konsole Abschnitt 3.6 ansprechbar ist.
  Der Name wird **nicht** als Hostname gesetzt!

Des Weiteren existieren Kommandos zur Übergabe von Terminals an bestimmte Schnittstellen der UML, die dann als Konsolen benutzt werden können.
Als Standard öffnet jedes System beim Booten sechs Xterms als Login-Konsole, das Verhalten kann aber ganz den eigenen Vorstellungen angepasst werden. Genauere Informationen zu recht umfangsreichen Konfigurationsmöglichkeiten von Terminals und zu anderen möglichen UML-Parametern liefert [Car05].

3.6 Management

Mit Hilfe der Management-Konsole `uml_mconsole` ist es möglich einige grundlegende Management-Aktionen an einer laufenden UML-Maschine vorzunehmen. Sie ist im selben Paket enthalten wie etwa `uml_switch` und wird unter SuSE automatisch installiert. Mit `uml_mconsole` ist es z.B. möglich virtuelle Maschinen einzufrieren, herunterzufahren oder neu zu booten, ohne an dem jeweiligen System angemeldet zu sein. Desweiteren kann virtuelle Hardware im laufenden

Betrieb hinzugefügt bzw. entfernt werden, was sich im Wesentlichen auf Netzwerkkarten und Festplattenimages beschränkt. Die Steuerung läuft ähnlich wie bei `uml_switch` über einen Unix-Socket. Generell ist das Tool allerdings nicht sehr leistungsfähig. Das ist auch der Grund warum diverse andere Management-Werkzeuge für UML existieren. Leider sind diese Neuentwicklungen meist nicht besser als das Original. Für die Praxis haben sie daher wenig Bedeutung.

Hier eine genaue Liste der Möglichkeiten.

- `(mconsole) version` - liefert die Versionsnummer des UML-Kernel.
- `(mconsole) halt / reboot` - beendet eine UML. Wirkt wie „Netzstecker ziehen" und hat eigentlich immer ein inkonsistentes Dateisystemimage zur Folge.
- `(mconsole) config` - fügt neue Hardware im laufenden Betrieb hinzu.
- `(mconsole) remove` - entfernt vorhandene Hardware im laufenden Betrieb.
- `(mconsole) sysrq` - ruft einen sysrq-Befehl auf.
- `(mconsole) cad` - sendet Ctl+Alt+Del an die UML.
- `(mconsole) stop` - pausiert eine UML.
- `(mconsole) go` - weckt sie wieder auf.
- `(mconsole) log` - schreibt Text in den Kernellog der UML.
- `(mconsole) proc` - holt Informationen aus dem /proc-Filesystem der UML.

Ein umfassendes Management ist mit `uml_mconsole` nicht machbar. In der Praxis verwendet man auch bei virtuellen Maschinen `ssh` zum Remote-Login und konfiguriert diese dann auf Kommandozeilenebene. Einzige Voraussetzung hierfür ist eine funktionierende Netz-Verbindung zum Managementinterface. Ist diese Voraussetzung nicht erfüllt, so hilft auch die Managementkonsole nichts. In diesem Fall muss man sich direkt am Host an einem Terminal der UML anmelden und den Fehler beheben.

Kapitel 4

Einsatz von UML

4.1 Arbeiten auf den virtuellen Maschinen

Prinzipiell bieten sich zwei grundsätzlich unterschiedliche Varianten an um auf einer entfernten UML zu arbeiten:
Login per `ssh` bzw. `ssh -X` auf der UML selbst, oder Login auf dem Host mittels `ssh -X` und anschließend (eventuell auch automatisiert) von dort das UML-System starten. Der erste Ansatz unterscheidet sich vom zweiten in einem sehr wesentlichen Punkt:
Ansatz eins geht davon aus, dass alle UMLs ständig laufen, Ansatz zwei davon, dass bei Bedarf ein virtuelles System gestartet wird. Das heißt, Ansatz eins ist Memory lastig, Ansatz zwei CPU und I/O lastig.
Es gibt jedoch noch einen weiteren Unterschied: wenn ein System schon läuft, muss es auch erreichbar sein. Man benötigt also zusätzlich zum im Praktikum verwendeten Netzwerk-Interface ein Management-Interface über das man die Systeme konfigurieren kann. Das entfällt bei der Boot-on-Demand Variante, da dort die Konsole der UML direkt zum User exportiert wird. Will man jedoch grafische Anwendungen aus dieser Konsole starten, so muss man auch hier zuvor das Display manuell exportieren. Ein Nachteil dieser Methode ist jedoch sicher, dass ein Login auf dem Host möglich sein muss. Das kann eventuell zum Sicherheitsproblem werden, da Dateisystem-Images der einzelnen Praktikumsteilnehmer untereinander nicht zugänglich sein sollen.
Anmelden kann man sich im Prinzip von jedem an das Netz angebundenen PC mit installiertem SSH-Client. Zur Nutzung von grafischen Programmen via `ssh -X` benötigt der Client jedoch zwingend einen laufenden X-Server. Dies sollte bei der Verwendung eines Linux-Clients, wie sie z.B. im CIP-Pool vorhanden sind, kein Problem darstellen; bei der Verwendung eines Windows-Rechners hingegen schon, da Windows ein anderes Grafik-Subsystem verwendet als Linux und Linux-Fenster damit nicht einfach auf Windowsoberflächen gezeichnet werden können. Dieses Hinderniss lässt sich jedoch mit Hilfe von `cygwin` [cyg05], das einen X-Server bereitstellen kann, beheben. Genauere Informationen liefert die Projekt-Homepage von `cygwin`. Eine andere Variante ist das Steuern von einem ebenfalls virtuellen Rechner, der hier z.B. von VMWare bereitgestellt werden kann und unter Linux läuft. Mit diesen Varianten kann man unter Windows weiterarbeiten ohne auf grafische Oberflächen verzichten zu müssen. Beide Möglichkeiten erfordern aber eine Installation und für VMWare fallen zusätzlich Lizenzgebühren an. Hinzu kommt, dass auch für das Einrichten eines an sich recht benutzerfreundlichen VMWare Grundkenntnisse über virtuelle Maschinen und Netzwerke von Nöten sind.

4.2 Minimale Konfiguration

Auf der vorhandenen Testumgebung (Pentium 4 3GHz, 1024MB RAM) benötigt ein virtuelles SuSE 9.3 „minimales System + X11" im Leerlauf ohne Login ca. 45MB Speicher, mit Login sind es ca. 65MB. Startet man dann noch einen virtuellen Windowmanager so sind es schon 100MB. Immer unter der Vorraussetzung, dass das System nichts zu tun hat. Arbeitet man aktiv mit einer Maschine, so braucht sie selbst bei scheinbar harmlosen Systemaufrufen signifikant mehr Speicher. Die CPU-Last, die eine virtuelle Maschine verursacht, hält sich dagegen in Grenzen. Im Leerlauf sind es maximal 0,5% ohne Windowmanager und ca. 1% mit. Aus diesen Werten ist bereits ersichtlich, dass weniger als 64MB RAM bei einer virtuellen Maschine zum ständigen Auslagern des Hauptspeichers auf die Platte führen würde. Auch SuSE gibt als Vorraussetzung für die Installation mittels YAST2 seit langem schon mindestens 64MB vor. Bei weniger RAM startet nur der textbasierte YAST. Die Verwendung von virtuellen Windowmanagern kommt ohnehin aus bereits erläuterten Problemen nicht in Betracht. Als minimale Konfiguration für den Arbeitsspeicher einer UML gelten somit 64MB.
Ein anderer zu analysierender Punkt ist die Verwendung von Swap-Files. Grundsätzlich benötigt man nicht zwingenderweise Swap-Files. Ist der einer UML zugewiesene Speicher jedoch erschöpft, so kann es ohne zusätzlichen Auslagerungsspeicher zum Einfrieren der virtuellen Maschine kommen. Insofern muss man sich entscheiden, ob man lieber wenig Arbeitsspeicher an die virtuellen Maschinen vergibt und jedem System separat ein Swap-File zuweist oder den Maschinen mehr Arbeitsspeicher zuteilt und auf Swap-Files verzichtet. In Summe bleibt die Auslastung des Arbeitsspeichers im Host dieselbe, da Swap-Files genau wie virtueller Arbeitsspeicher der UMLs im RAM des Hosts verbleiben. Da der Arbeitspeicher der virtuellen Maschinen im Host nicht sofort belegt wird, sondern ähnlich wie bei COW-Files erst, wenn er wirklich benötigt wird, ist die Variante ohne Swap-Files klar die einfachere, da die Konfiguration der Swap-Files entfällt. Leider wird einmal allokierter Speicher von Linux und somit auch von den UMLs nicht mehr freigegeben. Auf das Swap-File des Hosts sollte man allerdings nicht verzichten.
Die absolute Minimalkonfiguration für das Praktikum sind 64MB Arbeitsspeicher, um alle Aufgaben bearbeiten zu können. Für einige Aufgaben wie Arbeiten mit Yast2 oder Kompilieren von Sourcecode ist mehr Arbeitsspeicher hilfreich, jedoch nicht zwingend notwendig.

4.3 Anforderungen an den Host

User Mode Linux läuft solange flüssig, solange genug freier Arbeitsspeicher im Host vorhanden ist. Probleme treten in der Praxis erst auf, wenn der Speicher im Host knapp wird und einzelne Bereiche auf Platte ausgelagert werden müssen. Die Platte ist in diesem Fall dauerhaft ausgelastet und ein flüssiges Arbeiten auf den UMLs ist nicht mehr möglich.
Das Booten einer einzelnen UML-Instanz benötigt auf oben genanntem Testsystem, wenn sonst keine anderen Prozesse aktiv sind, ca. 40 Sekunden, fährt man mehrere Systeme gleichzeitig hoch, dauert es pro Instanz geringfügig länger und das System reagiert nur noch sehr träge auf Benutzerinteraktion. Dies liegt haupsächlich an der ständig ausgelasteten IDE-Festplatte. Ein leistungsfähiges SCSI-Raid zugrundegelegt, dürfte dieses Probem jedoch wenigstens ansatzweise beheben. Ein geeignetes Testsystem stand leider nicht zur Verfügung.
Die wichtigste Systemressource für den Host ist RAM und für den Fall, dass wirklich etwas in den SWAP Bereich ausgelagert werden muss, eine leistungsfähige Platte. Die CPU des Testsystems war bis auf einige extreme Ausnahmesituationen nie wirklich der Flaschenhals des Systems. Im Testfall

liefen 15 virtuelle Maschinen mit jeweils 128 MB RAM und SWAP sowie einem virtuellen Netzwerkinterface. Software-seitig liefen alle von SuSE installierten Standard-Dienste, allerdings keine grafische Oberfläche. Mit dieser Konfiguration war der Hauptspeicher des Testsystems erschöpft und das System begann den SWAP-Speicher zu nutzen, was man in der Praxis aus Performance-Gründen möglichst vermeiden sollte.
In dieser Konfiguration wurde auf fünf Maschinen gleichzeitig damit begonnen die Kernelquellen zu übersetzen, was das System stark beanspruchte. Erst nach dem Schließen einiger „arbeitsloser" virtueller Maschinen lief das System deutlich performanter. Zu Abstürzen der virtuellen Maschinen kam es während der Tests kein einziges Mal.
Zum Zeitpunkt der Tests lief User Mode Linux nur auf 32Bit-Systemen stabil. Ein Test auf einem zusätzlichen 64Bit Testrechner verlief negativ. User Mode Linux startete gar nicht erst. Genauere Recherchen ergaben, dass sich diese Version zum Zeitpunkt des Tests im Entwicklungsstadium befand und noch einige Fehler aufwies.
Geht man von einem maximal 4GB großem Adressraum des Arbeitsspeichers auf 32Bit-Umgebungen aus und behält 1GB für das zugrunde liegende Betriebssystem und etwas Puffer, so kann man 3GB Arbeitsspeicher für die virtuellen Maschinen einplanen. Bei über 40 benötigten Maschinen ergeben sich ca. 70MB Arbeitsspeicher pro UML, was nahe an der minimalen Konfigurations-Grenze liegt. Als Server für das Szenario sollte daher theoretisch ein Dual-Xenon Rechner mit 4GB RAM und SCSI-Raid oder vergleichbar ausreichen. Allerdings befindet sich das System dann schon in der Nähe seiner Belastungsgrenze.
Auf der Projekthomepage von User Mode Linux wurden vor kurzen Berichte verfügbar, die von ersten Erfolgen auf 64Bit-Architekturen berichten. Die erste funktionierende Version soll angeblich in der Kernelversion 2.6.12-rc4 enthalten sein. Aus diesem Grund ist es eine Überlegung wert, das Projekt versuchsweise auf einem 64Bit Serversystem auszutesten. Das bringt zwei Vorteile: 64Bit Prozessoren haben einen leichten Geschwindigkeitsvorteil gegenüber gleichwertigen 32Bit Prozessoren. Der weit größere Vorteil ist aber die Möglichkeit, mehr als 4GB RAM zu benutzen und damit den Flaschenhals des Systems zu weiten. Das Risiko, eine nicht für das Projekt geeignete Serverhardware zu kaufen ist nicht vorhanden, denn Linux lässt sich auch auf 64Bit-Umgebungen im 32Bit-Modus installieren. Als Serverhardware bietet sich in diesem Fall zum Beispiel ein Dual Opteron System mit 4GB RAM und SCSI Raid an. Bei voller Funktionsfähigkeit des Systems im 64 Bit Modus kann man dann den Speicher nach ersten Erfahrungswerten aus der Praxis beliebig auf zum Beispiel 6GB oder 8GB aufrüsten.

Kapitel 5

Schlussfolgerungen & Bewertung

Die Tests mit User Mode Linux in dieser Arbeit zeigen ganz deutlich, dass eine Umsetzung der Praktikumsszenarien grundsätzlich kein Problem darstellt. Bei allen Tests, angefangen beim Routing über tcpdump bis zum Compilieren der Kernelquellen trat kein einziger User Mode Linux spezifischer Fehler auf, nachdem alles ordungsgemäß konfiguriert war. Diese Stabilität und Robustheit der virtuellen Maschinen beindruckte besonders. Es kam nicht ein einziges mal zum Abstrurz eines UML Kernels. Allerdings dauerte es anfangs einige Zeit, bis wirklich alle Hürden beseitigt worden waren. Nahezu jede einzelne, der während dieser Arbeit vorgestellten Konfigurationsmöglichkeiten, ist in der Lage diese Harmonie im Zusammenspiel der einzelnen Komponenten zu zerstören. Besonders gilt das für die Erstellung des virtuellen Dateisystems.
Je nach Arbeitsverhalten der Teilnehmer und Auftreten von kurzzeitig hoher Rechnerlast könnte es bei Verwendung einer 32Bit-Umgebung notwendig sein, die benötigten virtuellen Maschinen auf zwei verschiedene Server zu verteilen um das Arbeiten an den Maschinen bei Lastspitzen nicht zur Geduldsprobe werden zu lassen. Jeder der beiden Server müsste dann unabhänging vom jeweils anderen die vorgestellten Szenarien mit je 20 Maschinen simulieren können. Das Praktikum liefe dann in zwei parallelen Gruppen ab. Alternativ könnte auch jeder der Server einen Teil eines erweiterten Szenarios bereitstellen. Dies hätte jedoch eine leichte Abwandlung im Kern des Netzwerkaufbaus zur Folge, da das zusätzliche Routing zwischen den beiden Hosts nicht völlig transparent umgesetzt werden kann.
Bei der Verwendung eines 64Bit-Systems als Host sollte das Szenario aber problemlos auf einem Server ausführbar sein. Voraussetzung hierfür ist lediglich ein erfolgreicher Test vom UML auf dieser Plattform. Nach Berichten von Jeff Dike, dem Hauptverantwortlichen Entwickler von User Mode Linux, sind solche Test in Zukunft zu erwarten.
Die Realisierung des „secservers" ist durch die Art der Umsetzung mit User Mode Linux nicht vorgeschrieben. Er kann entweder ebenfalls als virtueller Server bereitgestellt werden oder aber die von ihm bereitgestellten Dienste müssen einfacherweise vom Host bereitgestellt werden. Der zweite Ansatz klingt einfacher, da es recht umständlich erscheint, die zur Installation von Software benötigten SuSE CD/DVD-Images im Image eines virtuellen Dateisystems abzulegen um sie anschließend via NFS bereitstellen zu können. Dies lässt sich auf dem Host elleganter und vor allem effizienter lösen. Aufgrund des sehr Resourcen lastigen Bootens einer virtuellen Maschine, erscheint es angebracht, den zusätzlichen Aufwand für die Konfiguration der Managementinterfaces auf sich zu nehmen und im Gegenzug die virtuellen Maschinen durchlaufen zu lassen. Dies hätte zusätzlich den Vorteil, dass das komplette Rechnernetz ständig verfügbar ist, und man nicht mehr darauf angewiesen ist, dass alle Praktikumsteilnehmer zum selben Zeitpunkt arbeiten um das komplette Netz verfügbar zu haben.

Ein weiterer Vorteil bei dieser Variante ergibt sich dadurch, dass kein Login auf dem Host nötig ist um die virtuelle Maschine zu erreichen. Ein Login auf der virtuellen Maschine genügt. Selbst wenn die UMLs auf ihren Management-Interfaces keine öffentlichen IP-Adressen erhalten sollen, ist dieses Vorgehen möglich. Denkbar wäre z.B. ein Login-Server, von dem man Zugriff auf die UMLs hätte. Dieser ließe sich sogar virtuell implementieren. Noch einfacher wäre wahrscheinlich ein simples ssh-Port-Forwarding. So könnte man z.B. den Port 2211 des Hosts auf den ssh-Port 22 der virtuellen Maschine 11 weiterleiten. Jeder Praktikumsteilnehmer hätte so seinen eigenen Port, den man auch überwachen könnte um die „Arbeitsintensität" der einzelnen Studenten zu protokollieren. Nicht desto Trotz bleiben für eine endgültige Realisierung des Projekts noch einige Aufgaben zu bewältigen.
Es muss/müssen

- ein Konzept zum Aufbau des Management-Netzes erarbeitet werden, so dass die Teilnehmer im CIP-Pool oder von zu Hause aus an den Maschinen arbeiten können.
- eine Firewall implementiert werden, die zwar eine Konfiguration per SSH zulässt, ansonsten aber jegliche Komunikation der UMLs über das Managementinterface verbietet.
- eine Möglichkeit für die Teilnehmer geschaffen werden abgestürzte Maschinen neu zu starten.
- ein Tool zum Umschalten der Szenarien entwickelt werden, eventuell basierend auf VNUML[vnu05]. VNUML wird an der Universität von Madrid entwickelt und ist in der Lage komplexe virtuelle Netze hochzufahren. Dazu bootet es anhand einer XML-Konfigurationsdatei die benötigten Maschinen mit angegebener virtueller Hardware und erstellt die virtuellen Netze. Zugleich ist VNUML in der Lage einfache Konfigurationen an den einzelnen Maschinen vorzunehmen. Hierzu gehört etwa das Zuweisen von IP-Adressen an bestimmte Interfaces.
- die Kompatibilität von User Mode Linux auf 64Bit-Prozessoren getestet werden.
- ein Sicherungs- und Backupkonzept erarbeitet werden.
- die Aufgaben des Praktikums an die neue Umgebung angepasst werden.

Diese Aufgaben sprengen jedoch den Rahmen eines FoPras wie diesem. Zielsetzung dieser Arbeit war von Anfang an eine Machbarkeitsanalyse des Vorhabens. Zur Umsetzung der genannten Aufgaben und einer endgültigen Installation der Szenarien auf einem Serversystem bietet sich eine weitere Arbeit an, die auf den Erkenntnissen dieses Praktikums aufbauen kann. Im Rahmen einer Diplomarbeit scheint eine Realisierung denkbar.

Literaturverzeichnis

[boc05] *Bochs Hompage*.
http://bochs.sourceforge.net, 16.09.2005.

[Car05] James F. Carter. *Installing User Mode Linux on SuSE*.
http://www.math.ucla.edu/~jimc/documents/uml-install-suse.html, 31.01.2005.

[cyg05] *Cygwin Homepage*.
http://www.cygwin.com, 16.09.2005.

[Gru05] Roland Gruber. *Ansaetze für Betriebssysteme der Zukunft: User Mode Linux*.
http://www13.informatik.tu-muenchen.de/lehre/seminare/WS0405/hauptsem/Au
27.01.2005.

[Kno03] Gerd Knorr. *User Mode Linux with SuSE 9.0*.
http://www.suse.de/~kraxel/uml/howto.html, 28.10.2003.

[ska05] *SKAS-Patch*.
http://www.user-mode-linux.org/~blaisorblade/, 16.09.2005.

[uml05] *User Mode Linux Hompage*.
http://user-mode-linux.sourceforge.net, 16.09.2005.

[vir05] *MS Virtual PC Hompage*.
http://www.microsoft.com/windows/virtualpc/default.mspx, 16.09.2005.

[vmw05] *VMware Hompage*.
http://www.vmware.com, 16.09.2005.

[vnu05] *VNUML*.
http://jungla.dit.upm.es/~vnuml/, 16.09.2005.

[xen05] *Xen Hompage*.
http://www.cl.cam.ac.uk/Research/SRG/netos/xen, 01.09.2005.