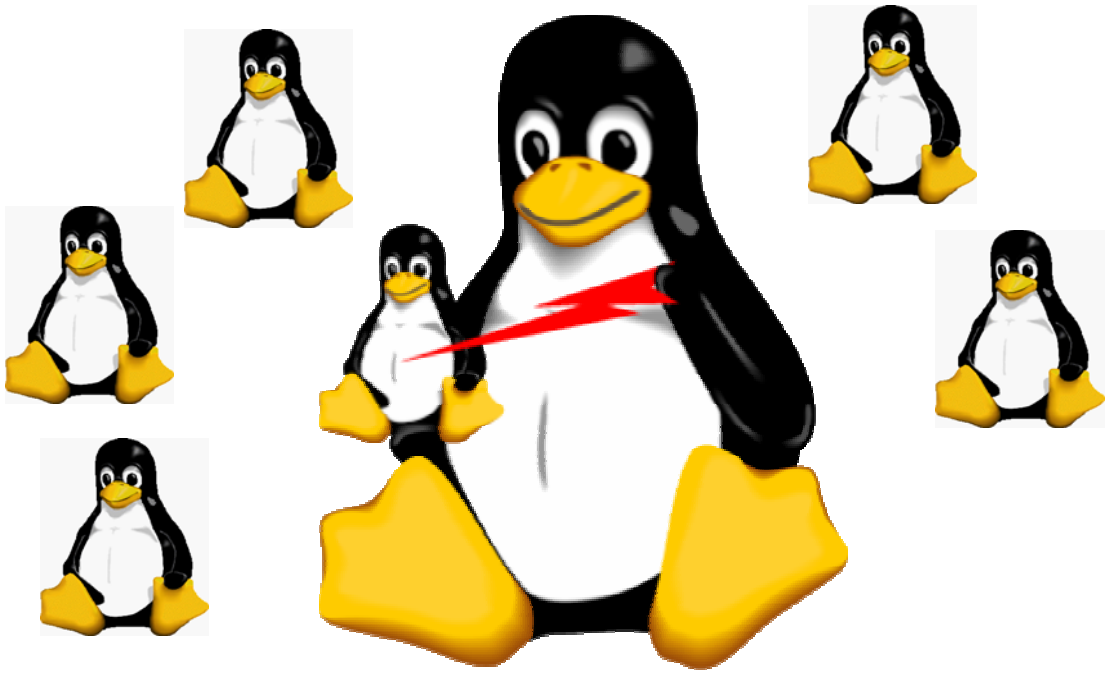
Projektarbeit

Virtuelle Hosts unter User Mode Linux



Dozent: Prof. Dr. Andreas Steffen

Partnerfirma: Netzschmiede GmbH
Rosenstrasse 14
8400 Winterthur

Kontaktperson: Herr S. Moser

Studenten: Martin Gonzenbach
Sandro Müller

Inhaltsverzeichnis

1. Management Summary

User Mode Linux (UML) is an open source project who's goal is to have virtually more then one kernel instance that could be started and run. With ones own filesystem and virtual network device one can in this way virtualizes a complete network with computers, switches and hubs. User Mode Linux offers specially for kernel programmers, linuxgeeks and more and more interesting applications for hosting companies.

The company Netzschmiede GmbH wants to offer there customers such UML-instances as virtual servers. It was our task to frame a project to find out how and if UML with the new 2.6 Kernel could be applied for web hosting and how such a system could be made to operate. Main aspects of the tasks are therefore backup, security, filesystems and network connections. While working on the tasks with different filesystems it was very quickly observed that, the choice of the filesystems plays a very important role for the future application, and therefore almost certainly requires the introduction of a journaling filesystem.

When taking about web hosting and customer service security should not be forgotten. For this reason various security aspects will be dealt with in a separate capture. Doing so, we have also noticed, that there are limits and sometimes compromises have to be made. On one hand it would be comfortable to operate a central patch management. On the other hand this can only be done efficiently when each person has the root access for all machines. In addition such an approach hides a danger that after an update specific customer projects can not be run anymore.

Another difficulty is the configuration of the network. The TUN/TAP driver is for web hosting a save and suitable solution. The virtual computers are therefore directly from the internet trough the public IP address available.

Finally, various projects for administrative and management purposes were examined by UML instance. Unfortunately, all such tools tested show deficiencies specially by management of 2.6 kernel UML and the function range.

To summarise the above, User Mode Linux offers interesting applications and is a very useful project. However, numerous pitfalls have to be overcome.

2. Zusammenfassung

Bei User Mode Linux (UML) handelt es sich um ein Opensourceprojekt welches zum Ziel hat, auf einem Linuxrechner mehrere virtuelle Kernelinstanzen starten und laufen zu lassen. Mit einem eigenen Filesystem und virtuellen Netzwerkdevices lassen sich auf diese Art komplette Netzwerke mit Rechnern, Switches und Hubs virtualisieren. User Mode Linux bietet insbesondere für Kernelentwickler, Linuxgeeks und je länger desto mehr auch für Hostingfirmen interessante Einsatzzwecke.

Die Firma Netzschmiede GmbH möchte solche UML-Instanzen als virtuelle Server ihren Kunden anbieten. Unsere Aufgabe war es im Rahmen einer Projektarbeit herauszufinden ob und wie sich UML mit dem neuen 2.6er Kernel für das Webhosting einsetzen lässt und wie solche Systeme betrieben werden könnten. Wesentliche Aspekte dieser Arbeit sind somit Backup, Sicherheit, Filesysteme und Netzwerkanbindungen. Während der Arbeit mit verschiedenen Filesystemen konnte rasch festgestellt werden, dass die Wahl des Filesystemtyps eine sehr entscheidende Rolle für den künftigen Einsatz mit UML spielt und darum fast zwingend ein Journalingfilesystem eingesetzt werden muss.

Wenn es um Webhosting und Kundenservices geht, darf das Thema Sicherheit nicht vernachlässigt werden. Es werden deshalb diverse Sicherheitsaspekte in einem separaten Kapitel behandelt. Dabei haben wir aber auch gesehen, dass es Grenzen gibt und manchmal Kompromisse eingegangen werden müssen. So wäre es auf der einen Seite komfortabel ein zentrales Patchmanagement zu betreiben. Anderseits kann dies nur effizient realisiert werden, wenn eine einzelne Person auf allen virtuellen Maschinen Root Zugriff hätte. Zudem birgt ein solches Vorgehen die Gefahr, dass nach einem Update allenfalls spezifische Kundenprojekte nicht mehr lauffähig wären.

Ein weiterer Knackpunkt ist die Konfiguration des Netzwerkes. Der TUN/TAP Treiber stellt für Webhosting eine sichere und geeignete Lösung dar. Die virtuellen Rechner sind damit vom Internet direkt über ihre öffentliche IP Adresse erreichbar.

Zuletzt wurden verschiedene Projekte zur Verwaltung und Administration von UML-Instanzen untersucht. Leider weisen noch alle getesteten Programme wesentliche Mängel

auf, insbesondere bei der Verwaltung von 2.6er Kernelinstanzen und dem Funktionsumfang.

Kurz gesagt bietet User Mode Linux interessante Einsatzgebiete und ist ein sehr spannendes Projekt, wobei aber doch die einen oder anderen „Pitfalls“ zu überwinden sind.

3. Einleitung

3.1. Aufgabenstellung

Projektarbeiten im Sommersemester 2004 - PAKI2 Sna01

Virtuelle Hosts unter User Mode Linux

Studierende:	Martin Gonzenbach, KI3b
	Sandro Müller, KI3b
Partnerfirma:	Netzschmiede GmbH, Rosenstrasse 14,
	8400 Winterthur
	http://www.netzschmiede.ch
Termine:	Ausgabe: Dienstag, 9.03.2004, 17.00 Uhr im E509
	Abgabe: Freitag, 2.07.2004

Beschreibung:

Mit User Mode Linux (UML) kann der Linux-Kernel als Prozess im Userspace gestartet werden. Dies erlaubt auf einem einzigen Rechner mehrere virtuell vernetzte Linux-Systeme laufen zu lassen. Ab Version 2.6 ist User Mode Linux ein fester Bestandteil des Linux-Kernels.

Im Rahmen dieser Projektarbeit soll User Mode Linux eingesetzt werden, um eine gewünschte Anzahl virtueller Systeme "auf Knopfdruck" konfigurieren zu können. Die virtuellen Systeme sollen bestimmte Funktionen im realen Netzwerk übernehmen können (z.B. Webserver, VPN-Gateway).

Aufgaben:

- Verstehen und Aufsetzen von User Mode Linux (Debian)
- Herstellung von Tools zur Verwaltung der virtuellen Systeme
- Optimierung der Konfiguration hinsichtlich Ressourcenverbrauch auf dem Hostsystem
- Betriebskonzept der Server-Infrastruktur (z.B. Einspielen von Security-Patches)

Infrastruktur / Tools:

Raum: E523

Rechner: 2 Rechner unter Debian Linux

SW-Tools: User Mode Linux

IP-Adressrange: 160.85.191.0 - 160.85.191.16

Literatur / Links:

User-Mode Linux Home Page	http://user-mode-linux.sourceforge.net
User-Mode Linux Community Site	http://usermodelinux.org
Debian Home Page	http://www.debian.org

Winterthur, 8. März 2004

Prof. Dr. Andreas Steffen

3.2. Erklärung betreffend das selbständige Verfassen der Projektarbeit im Dept. T

Mit der Abgabe dieser Projektarbeit versichern die Studierenden, dass sie die Arbeit selbstständig und ohne fremde Hilfe verfasst haben. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Die unterzeichnenden Studierenden erklären, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 35 und 36 (Unredlichkeit und Verfahren bei Unredlichkeit) des Reglements für Prüfungen am TWI sowie die Bestimmungen des Disziplinarverfahrens der Hochschulordnung in Kraft.

Ort, Datum: Unterschriften

....
Martin Gonzenbach

.....
Sandro Müller

3.3. Abgrenzung

User Mode Linux ist ein sehr grosses Gebiet, weshalb wichtig ist die vorliegende Projektarbeit sauber abzugrenzen.

Für die Projektarbeit berücksichtigt

Als Betriebssystem für den Host, wie auch für die UML Instanzen muss die Debian Linux Distribution verwendet werden.

Für den Hostkernel sowie auch für den UML-Kernel soll ein 2.6er Kernel verwendet werden. Der betreuende Dozent, Herr Steffen hat uns beim Start der Arbeit mitgeteilt, dass der Einsatz eines 2.6er Kernel uns vor Probleme stellen könnte. Aus diesem Grunde wurde vereinbart, mit dem 2.4er Kernel zu arbeiten, falls wir mit dem 2.6er Kernel nicht zum Ziel kommen.

Auf den UML Instanzen sollen Apache Webserver mit einfachen HTML oder PHP Testseiten laufen.

Für die Verwaltung der UML-Instanzen wollen wir verschiedene, vorhandene Tools auf deren Tauglichkeit testen. Zur Verwaltung eigener UML-Instanzen möchten wir ein Shellskript erstellen, mit welchem eine UML gestartet und gestoppt werden kann.

Für die Projektarbeit nicht berücksichtigt

In diesem Dokument werden wir nicht beschreiben, wie Debian auf einem x86-Rechner installiert wird. Dies kann auf der Debianhomepage[1] nachgelesen werden. Unsere Informationen haben wir auch von dort bezogen.

Grundsätzlich ist UML nicht auf eine bestimmte Linux Distributionen beschränkt. Für diese Projektarbeit wird aber ausschliesslich mit der Debian Distribution gearbeitet.

[1] http://www.debian.org

3.4. Die Firma Netzschmiede GmbH

Ansprechpartner

Herr Simon Moser
Netzschmiede GmbH
Rosenstrasse 14
8400 Winterthur
simon.moser@netzschmiede.ch

Firma Netzschmiede GmbH

Auf der Homepage der Firma Netzschmiede steht unter Webhosting folgendes:

Netzschmiede deckt die Bedürfnisse von wirtschaftlichem Mail Hosting bis zu dynamischen Internetauftritten zu günstigen Konditionen ab. Das Angebot deckt die hohen Anforderungen von Firmenkunden ebenso ab wie die Bedürfnisse von Privatpersonen. Unsere Internet-Plattform bietet ihnen eine stabile Grundlage für die Planung und Realisierung ihrer Internet-Projekte.[2]

---

[2] http://www.netzschmiede.ch

3.5. Was ist nun eigentlich UML genau?

UML ist eine Abkürzung für **U**ser-**M**ode-**L**inux. Dabei handelt es sich um ein Open Source Produkt mit der Möglichkeit auf einem Rechner mehrere virtuelle Systeme laufen zu lassen. Diese sind völlig unabhängig voneinander, können übers Netzwerk mit einer eigenen IP erreicht werden und verschiedenen Distributionen entstammen.

Definitionen:	physikalischer Rechner:	Host
	virtuelle Rechner:	UML-Instanz, virtueller Host

Häufig wird UML mit den kommerziellen Produkten VMware oder MS VirtualPC verglichen. Dies ist nicht ganz abwegig, gleichzeitig aber auch nicht ganz korrekt. VMware unterstützt nämlich nicht nur Linux, sondern auch andere Betriebssysteme. So können als Host Betriebssystem fast alle aktuellen Windowsversionen und Linuxdistributionen verwendet werden. Darauf können Windows Betriebssysteme, verschiedene Linuxdistributionen, Novell Netware und FreeBSD emuliert werden. VMware emuliert im Gegensatz zu UML einen kompletten Rechner inklusive Bios.
Bei UML kann nur ein Linux auf einem Linux betrieben werden. Innerhalb dieser Einschränkung ist man dann aber doch sehr flexibel. So können auf dem physikalischen und dem virtuellen Host unterschiedliche Kernel und Distributionen laufen.
Jede virtuelle UML-Session benötigt ihr eigenes Filesystem und ist damit vollständig unabhängig von den anderen Sessions. Es gibt aber auch Möglichkeiten die Filesysteme zu „teilen“ (siehe COW Filesysteme).
Unter den diversen Hosts, welche man jetzt auf einer Maschine laufen lassen kann, gibt es die Möglichkeit ein virtuelles Netzwerk zu realisieren. Damit kann z.B. jedes System von aussen über eine eigene IP angesprochen werden. Der Netzwerkverkehr aller virtuellen Rechner läuft auf demselben physikalischen Netzwerkinterface des Hosts zusammen.
Das Einsatzgebiet für UML ist sehr vielfältig. So können Kernelentwickler mittels UML ihre Kernel zu Testzwecken debuggen. Um ganze Netzwerke zu simulieren und damit

beispielsweise ein Firewall- oder VPN-Konzept zu testen ist der Einsatz von UML besonders interessant. Selbst dafür wird heute UML als Honeypots verwendet.

3.6. Webhosting mit UML

Unsere Partnerfirma Netzschmiede hat die Idee UML für Webhosting einzusetzen. Der wesentliche Punkt dabei ist, dem Kunden nicht einen vordefinierten Dienst zur Verfügung zu stellen, sondern gleich einen ganzen virtuellen Linuxserver. Auf diesem Rechner kann er selber bestimmen welche Skripts und Dienste er einsetzen möchte. Diese Server werden nun eben mit UML virtuell simuliert.

Der Kunde kann auf seinem virtuellen Host alle von ihm gewünschten Skriptsprachen, Datenbanksysteme und Applikationen einsetzen. Alle anderen, auf derselben physikalischen Maschine laufenden UML-Instanzen sind von seiner Konfiguration nicht betroffen.

Für die meisten Leute ist aber einen eigenen physischen Webserver zu betreiben und diesen an den schnellen und sicheren Internet-Backbone des Providers anzuschliessen (Server Housing) doch zu teuer und zu aufwändig. Die Hardware- und Wartungskosten des Hosts werden durch alle UML-Kunden auf einem System berappt. So kann mit UML Webhosting dem Kunden zu einem vernünftigen Preis ein Produkt geboten werden, welches zwar häufig gewünscht wird, meist aber viel zu teuer ist.[3]

[3] Netzwoche Nr. 19, 12 Mai 2004, S. 8 – 9 und ct 1. Juni 2004 S. 142 ff.

3.7. Laborrechner (Server für UML)

Hardware (Laborrechner der ZHW)

Prozessor	Intel Pentium 4, 1800 MHz
Arbeitsspeicher	256 MB SDRAM Cache 512 KB
Harddisk	Wechseldisk Maxtor DiamondMax Plus 9, 60 GB
2x Ethernet	3Com3C905CX-TXM, 100 Mbit

Partitionierung der Festplatte (Laborrechner der ZHW)

Root Partition:	Primär	20 GB	Linux ext2
Swap Partition	Primär	3 GB	Swap
Rescue Partition (HD-Images)	Primär	20 GB	FAT 32

Hardware (Dellrechner von der Firma Netzschmiede GmbH)

Prozessor	Intel Pentium 4, 3,0 GHz HT
Arbeitsspeicher	1024 MB SDRAM Cache 1024 KB
Harddisk	80 GB
Ethernet	On Board 100Mbit

Partitionierung der Festplatte (Dellrechner von der Firma Netzschmiede GmbH)

Root Partition:		456MB	Linux ext3
Swap Partition		506 MB	Swap
/home Partition		37 GB	Linux ext3
/tmp		897 MB	Linux ext3
/usr		4.6 GB	Linux ext3
/var		2.8 GB	Linux ext3

Betriebssystem

Auf den Laborrechnern haben wir das Linux Betriebssystem von Debian installiert. Es handelt sich dabei um die Version Debian Sarge unstable.
Als Standardinstallationsquelle wählten wir folgende Reihenfolge: CDROM, http, ftp

Kernel / Patch	Version / Name
Hostkernel	2.6.3
Hostkernel Patch	Host-skas3—2.6.3-v1.patch
UML Patch	Kernelsource 2.6.4/7 und uml-patch-2.6.4-1 und uml-patch-2.6.7-01

Wir einigten uns, der Einfachheit halber, immer dasselbe Passwort zu verwenden. Wohl wissend, dass eine solcher Umgang mit Passwörtern in einer produktiven Umgebung fatale Folgen haben könnte. Das Passwort lautet **„razorback“**.

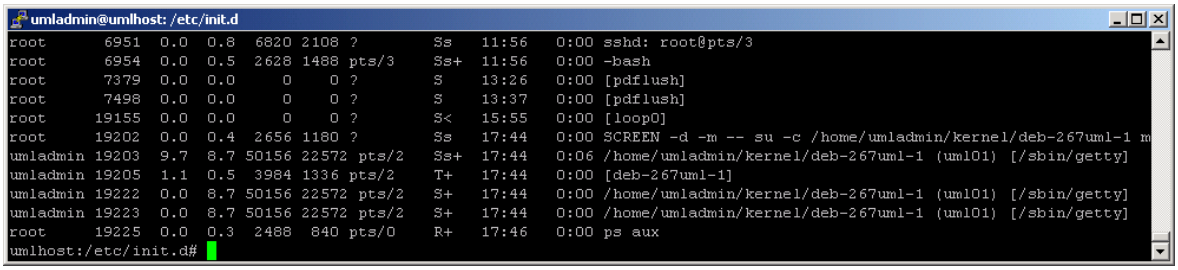
4. Konzepte und Architektur

4.1. SKAS- und TT- Mode

Zu Beginn der Entwicklung des User Mode Linux wurde jeder Prozess einer UML-Instanz als eigener Prozess in dem Host abgebildet. Daraus resultiert, dass der Administrator des Host-Systems jeden einzelnen Prozess von allen UML-Instanzen sehen kann. Dazu kommt, dass Prozesse standardmässig Schreibrechte für den Kernel besitzen. Somit kann also von einer UML-Instanz aus der Host manipuliert werden. Dies kann natürlich je nach Einsatzgebiet ein grosses Sicherheitsproblem darstellen. Die Firma Netzschmiede ist interessiert daran, UML als Hostingservice anzubieten. Dabei wäre es natürlich verheerend, wenn ein Kunde von einer UML-Instanz auf den Host und damit auch auf die anderen Webserver (UML-Instanzen) zugreifen könnte.
Lösen kann man dieses Zugriffsproblem mit dem UML-Jail mode. Damit werden die UML-Daten nur noch in dem „read only" Modus gestartet. Leider ist mit diesem Hostpatch nur ein Teil des Problems gelöst, denn nach Anwendung dieses Patchs müssen grosse Performanceeinbussen in Kauf genommen werden.
Die Entwickler von User Mode Linux setzten sich aber seit Ende 2002 intensiv mit dem Thema Sicherheit auseinander und erstellten weitere Hostpatches, um die erwähnten Zugriffs und Performance Probleme in den Griff zu kriegen.
Der „alte" Modus wird nun TT-Mode, für Tracing Thread genannt und der neue Modus bezeichnet man als SKAS-Mode. SKAS steht für Separate Kernel Address Space.
Der SKAS-Patch wird auf die reine Kernelsource für ein Hostsystem angewendet. Weiter muss in der Hostkernelconfig die Option „proc mm" und in der UMLkernelconfig die Option „Separate Kernel Address Space support" aktiviert werden. Ob diese Voraussetzungen erfüllt sind, erfährt man durch folgende Kernelmessages beim starten einer UML-Instanz.

```
Checking for the skas3 patch in the host...found
Checking for /proc/mm...found
```

Im Gegensatz zum TT-Mode sind beim SKAS-Mode auf dem Host nur noch vier Prozesse für jede UML-Instanz sichtbar., Mit dem Befehl „ps aux“ auf dem Hostrechner, kann dies wie im folgenden Screenshot dargestellt, nachvollzogen werden.



Screenshot mit den vier UML-Prozessen

Die vier Prozesse haben folgende Aufgaben:

- Der UML-Kernel Thread läuft im separierten Kerneladressraum, führt den Kernelcode aus und fängt die Systemcalls der UML ab.
- Der UML-Userspace Thread führt den Code von UML-Prozessen aus und wechselt zwischen dem Hostadressraum bei jedem Kontexswitch einer UML.
- Der ubd-Treiber Thread ist für das asynchrone IO zuständig.
- Der SIGIO-Emulator Thread schreibt.

Innerhalb der Instanzen wird nur noch mit Threads gearbeitet. Somit kann ein Hostadministrator nicht mehr sehen, was innerhalb einer UML am laufen ist. Der Hostadministrator kann natürlich weiterhin die Prozesse einer UML-Instanz killen, wodurch dann aber gleich die gesamte Instanz abgeschossen wird und nicht mehr nur Teile davon.

Die Vorteile des SKAS-Mode liegen nicht nur in den verbesserten Sicherheitsfeatures, sondern steigern insbesondere auch die gesamte Performance von Host und UMLs.

Trotz SKAS-Patch kann eine UML-Instanz im TT-Mode laufen, indem man den UML-Kernel mit der Bootoption „mode=tt“ startet.

Der Entscheid für den SKAS-Mode ist uns sehr leicht gefallen, überwiegen doch die Vorteile sehr stark gegenüber dem TT-Mode.

4.2. Speicherverwaltung

Kurz vorweg: Unsere Erfahrungen mit UML haben gezeigt, dass man auf dem Hostrechner fast nicht genügend physikalischen Speicher (RAM) zur Verfügung haben kann, denn je mehr RAM vorhanden sind, umso performanter laufen auch die verschiedenen UML-Instanzen, respektiv umso mehr UMLs können gleichzeitig laufen gelassen werden.
Jede UML erstellt standardmässig im Verzeichnis /tmp (Umgebungsvariable TEMPDIR) eine temporäre Datei mit derselben Grösse wie die UML-Instanz RAM alloziert hat (Bootoption mem=64M). Aus diesem Grund ist es empfehlenswert das tempfs-Filesystem für /tmp zu verwenden. Tempfs ist ein Dateisystem welches dynamisch, je nach Bedarf in einer wachsenden und schrumpfenden RAM-Disk residiert. Bei Bedarf kann diese Art von RAM-Disk ausgelagert (swapping) werden. Wird nun die RAM-Disk mit den temporären Dateien geswappt, liegen diese zwar wieder auf der Festplatte, verschwinden jedoch von dort wieder. Um tempfs zu verwenden, sollen Host und UML-Kernel mit der Option CONFIG_TMPFS=Y unter „virtual memory file system support“ konfiguriert und übersetzt werden. Danach kann ein solches tempfs entweder mit folgenden Befehlen erzeugt werden.

```
# mkdir /tmp/umlram
# mount –t tempfs –o mode=1777,size=512M none /tmp/umlram
```

Oder mit einer zusätzlichen Zeile

```
[ tempfs   /tmp/umlram   tmpfs   defaults,size=512M   0  0 ]
```

in der /etc/fstab permanent definiert werden. Die Grösse des tempfs kann den eigenen Bedürfnissen angepasst werden, sollte aber mindestens der Ramsumme aller UMLs entsprechen und nicht grösser sein, als der verfügbaren Speicher (RAM + SWAP) auf dem Host. Ein kleines Beispiel:

RAM Host:	1024MB
SWAP Host	1024MB
16 UML-Instanzen mit je 256MB RAM sind total	4096MB

Somit bräuchte der Host 2GB mehr Speicher (SWAP oder RAM)

Wird „ls /tmp/umlram“ ausgeführt, erscheint ein leeres Verzeichnis obwohl mehrere UML-Instanzen am laufen sind und somit mehrere Ramfiles ersichtlich sein müssten. Die Ramfiles können mit „`lsof | grep /tmp/umlram`“ sichtbar gemacht werden. Lsof ist ein Unixspezifiziertes Diagnosewerkzeug und steht für LiSt Open Files. Wie der Name eigentlich bereits aussagt, werden damit alle durch Prozesse geöffneten Dateien auf einem System angezeigt, so auch unsere UML-RAM.

4.3. Filesysteme: Root und Swap Partitionen

Für einen virtuellen Rechner braucht es nicht nur einen UML-Kernel, sondern man benötigt auch ein eigenes Rootfilesystem. Darin werden die spezifischen Daten (Distributionen) und Konfigurationen abgespeichert, welche von einem UML-Kernel zur Laufzeit benötigt werden. Dieses Rootfilesystem wird beim booten einer virtuellen Maschine als Parameter übergeben. Aus Sicht des Hostrechners handelt es sich bei einem solchen UML-Rootfilesystem lediglich um eine Datei, auch Loopbackdatei genannt, mit einer fixen Grösse. Durch geschicktes Anlegen dieser Datei, besitzt diese zwar eine feste Grösse, belegt aber effektiv nur so viel Speicherplatz wie Daten innerhalb der Loopbackdatei vorhanden sind. Innerhalb dieser fest definierten Grösse kann die Datei dynamisch wachsen. Dieser Dateityp wird auch Sparse File genannt. Mit dem Befehl „`ls –alhs`“ werden alle Angaben zur Datei (Dateigrösse und effektiv belegter Speicherbedarf) angezeigt. Wie bei einem Filesystem auf einer Partition, kann auch das Rootfilesystem für UML-Instanzen von beliebigem Typ sein (ext2, ext3, ReiserFS, etc.). Mit dem Befehl

```
# mount /Pfad/NameDesFilesystems /mnt –o loop
```

kann es direkt in das temporäre Mountverzeichnis des Hosts gemountet und dort bearbeitet werden. Auf diese Art können z.B. neu erzeugte Filesysteme konfiguriert werden ohne eine UML-Instanz zu starten.

Damit die Speicherbelastung des Hosts durch die einzelnen UML-Instanzen nicht allzu gross ist, können auch Swap Filesysteme erzeugt und wie das Rootfilesystem beim booten

als Parameter mitgegeben werden. Dies macht sowohl bei der parallelen Nutzung von vielen UML-Instanzen als auch bei der Nutzung von wenigen UMLs mit hohem Ressourcenverbrauch (Speicher) Sinn.

Stabilität und Tauglichkeit der verschiedenen Filesysteme

Für unsere ersten Versuche haben wir ein Rootfilesystem vom Typ ext2 eingesetzt. Dieses File haben wir nicht selber erzeugt, sondern von der UML-Webseite heruntergeladen. Die Stabilität, dieses Filesystems hat uns allerdings nicht besonders überzeugt. Wenn eine Instanz nicht ganz sauber mit dem Befehl „`shutdown -h now`“ heruntergefahren wurde, war danach das Filesystem korrupt und konnte meist auch mit einem „`fsck`“ nicht mehr repariert werden. Folglich konnten wir keinen UML-Kernel mehr mit diesem Filesystem booten.

Das ext2 Dateisystem ist ziemlich „träge“, was die Buchführung seiner Aktivitäten betrifft. Wird ein ext2-Dateisystem „gemountet“, vermerkt es nur, dass es in Benutzung ist. „Not clean“ lautet der Status des Systems. Folgt später dann ein „umount“, wechselt der Status zurück auf „clean“. Wenn das saubere Abhängen - aus welchem Grund auch immer – gescheitert ist, muss ext2 beim folgenden Systemstart alle Dateien überprüfen, um eventuellen Ungereimtheiten auf die Schliche zu kommen. An diesem Punkt setzt die Idee des Journaling Dateisystems an. In ihm werden alle momentan bearbeiteten Dateien protokolliert, so dass im Falle eines Systemcrashs beim Wideranlauf einzig diese Dateien auf Inkonsistenzen hin zu überprüfen sind. Eine weitere Massnahme ist die Einführung eines transaktionsbasierten Modells, wonach eine Datei solange ihre Gültigkeit behält, bis die Bearbeitung einer neueren Version vollständig abgeschlossen wurde. Somit wird die fehlerträchtige Zeitspanne auf den Moment des Abspeicherns reduziert.[4]

Aus diesen Gründen erstellten wir uns ein eigenes Filesystem von dem Typ ReiserFS mit der fixen Grösse von 500MB. Bei ReiserFS handelt es sich um eines der leistungsfähigsten und bekanntesten Journalingfilesystemen, wobei ext3 ähnliche Eigenschaften wie ReiserFS aufweist. Die Wahl kann somit nach den eigenen Vorlieben getroffen werden.

[4] http://www.linuxfibel.de/filesys.htm

Das ReiserFS Filesystem hat sich als wesentlich stabiler bewährt. Selbst wenn auf dem Host einfach der Prozess der UML-Instanz mit „`kill`“ abgeschossen wird, kann man den UML-Kernel sofort wieder mit dem Reiser Filesystem starten.
Die einzige Möglichkeit ein solches Filesystem zu zerstören ist, wenn das Filesystem vom Host aus gemountet wird, während eine laufende UML-Instanz darauf zugreift. Die UML-Instanz scheint zwar fehlerfrei weiterzulaufen. Wird diese aber neugestartet, kann sie nicht mehr mit diesem Filesystem hochfahren. Die Daten sind allerdings noch nicht ganz verloren, da das Filesystem auf dem Host weiterhin gemountet werden kann. Somit könnten die Daten manuell noch gerettet werden. Wir gehen allerdings davon aus, dass ein Hostadministrator über ein ausreichendes technisches Verständnis verfügt, um solche Dinge zu unterlassen ;-).

4.4. Copy on Write (COW)

Was ist COW

Bei der Verwendung von COW arbeiten alle UML-Instanzen mit demselben Rootfilesystem, welches für alle User schreibgeschützt ist. Diese Datei wird auch Backingfile genannt. Jeder UML-Instanz wird beim Start eine weitere Datei (COW-File) übergeben, in welche alle Änderungen des Rootfilesystem zurückgeschrieben werden. Die UML-Instanz holt sich somit alle Informationen aus diesem COW-File. Backingfile und COW-File spielen wie folgt zusammen: Das COW-File beinhaltet einen Header, in welchem eine Magicnummer zur Unterscheidung eines COW-Files von einem normalen Dateiimage steht, eine Versionsnummer, der Pfad zum Backingfile, Sektorgrösse der Datei sowie einen Zeitstempel mit der Grösse und dem letzten Änderungsdatum der Backingdatei[5]. Deshalb ist es sehr wichtig zu wissen, dass eine Backingdatei nie geändert werden darf, sofern man noch mit bestehenden COW-Files weiterarbeiten möchte. Mit COW realisierte Rootfilessysteme sind völlig unabhängig voneinander, was ein entscheidender Punkt

[5] http://www.usenix.org/publications/library/proceedings/als01/full_papers/dike/dike_html/node5.html

betreffend Sicherheit darstellt. Es bleibt gewährleistet, dass kein Besitzer einer UML-Instanz auf Informationen einer anderen Instanz zugreifen kann.
Sinnvoll ist der Einsatz von COW wenn mehrere UML-Instanzen gleichzeitig mit derselben Grundkonfiguration laufen sollen, oder der Diskplatz auf einem Hostsystem eine kritische Grösse darstellt, denn mit diesem Konzept kann wesentlich Festplattenplatz eingespart werden.
Wenn UML-Instanzen unter verschiedenen Distributionen laufen sollen, muss für jede Distribution ein eigenes Backingfile erzeugt werden. Innerhalb derselben Distribution können aber wieder mehrere Instanzen mit demselben Backingfile arbeiten.

Die Vorteile von COW in Bezug auf das Webhosting

Gehen wir davon aus, dass wir auf einem Hostsystem mehrere UML-Instanzen von verschiedenen Kunden am laufen haben. Jedem Kunden wird ein eigenes Rootfilesystem mit einer Grösse von 10GB zur Verfügung gestellt. Sind diese nun mit effektiv 500MB belegt, müssen aber trotzdem bei der Sicherung immer die gesamten 10GB gesichert werden. Während der Sicherung kann die UML-Instanz zwar laufen, muss aber in einen „schlafenden" Zustand gesetzt werden. Wenn ich nun 20-mal so viel Zeit, wie effektiv nötig wäre zum Sichern der Daten brauche, ist das ineffizient. Ein Webhost sollte aber möglichst geringe Downtime haben und rund um die Uhr verfügbar sein. Aus diesem Grund ist eine kurze Backupzeit sehr wertvoll. Wenn nun eine Sicherung einer gesamten UML-Instanz gemacht werden muss, reicht es einfach die COW-Datei zu sichern. Neuinstallierte Programme, Tools, Updates, Patches und Daten werden ebenfalls in die COW-Datei geschrieben. Für den Benutzer besteht somit kein Unterschied, zwischen einem eigenenständigen Rootfilesystem und einer COW-Datei ersichtlich.

Weitere Aspekte

Wie wir gesehen haben, ist dieses dynamische Wachsen nicht nur mit COW möglich, sondern lässt sich auch mit einem Sparsefile realisieren. Wenn ein eigens Rootfilesystem einer UML-Instanz nun effektiv keinen Platz für weitere Daten bietet, kann dies im Gegensatz zur COW-Variante problemlos vergrössert werden. Eine solche Vergrösserung sollte allerdings nie ohne vorheriges Backup der Datei durchgeführt werden, schlug doch

auch bei unserem Test dieser Vorgang einmal fehl. Bei COW, muss hingegen das Backingfile vergrössert werden, was zur Folge hat, dass danach alle darauf zugreifenden COW-Files neu erstellt werden müssen. Problemlos konnte hingegen ein COW-File zurück in das Backingfile geschrieben werden, d.h. mit dem „uml_moo“ Tool wird quasi ein neues Backingfile aus dem Inhalt des ursprünglichen Backingfile und des COW-Files erstellt. Somit muss sich ein UML-Hoster sehr genau überlegen mit welchem Verfahren er seinen Dienst anbieten möchte. Damit kostet Flexibilität vor allem viel Festplattenkapazität, welche allerdings heute ebenfalls recht preiswert zur Verfügung steht.

4.5. Hostfileaccess

Mounten des Hostfilesystems

Es gibt die Möglichkeit von einer UML-Instanz aus direkt ohne Netzwerkverbindung auf das Filesystem des Hosts zuzugreifen. Es kann mit einem einzigen Konsolenbefehl gemountet werden, vorausgesetzt die Option „Host filesystem“ ist im UML-Kernel aktiviert. Dies kann durch folgende Eingabe erfolgen:

```
UML# mount none /mnt –t hostfs
```

Nach erfolgreicher Ausführung des Befehls, kann unter /mnt von der UML-Instanz aus auf das gesamte Hostfilesystem zugegriffen werden, oder es wird die Meldung „mount: fs type hostfs not supported by kernel“ angezeigt.

Sicherheitsbedenken

Die einzige Methode, um diesen Zugriff zu unterbinden, ist darauf zu achten, dass der UML-Kernel wie erwähnt konfiguriert ist. Dieser Hostfileaccess mag ja sehr praktisch zum Beispiel für einen Kernelentwickler sein, hat aber in einer Multiuserumgebung, in welcher nicht jedem Benutzer 100% vertraut werden kann, unserer Meinung nach nichts zu suchen. Für die Firma Netzschmiede ist es deshalb notwendig darauf zu achten, dass die Option „Host Filesystem“ in den UML-Kerneln für die Hosting Kunden deaktiviert ist.

4.6. Virtuelles Netzwerk

Die verschiedenen Transporttypen

Im Zusammenhang mit UML werden auf der Projekthomepage sieben verschiedene Transporttypen empfohlen, um ein virtuelles Netzwerk auf einem Netzwerkdevice zu implementieren.

- Ethertap
- TUN/TAP
- Multicast
- Switch Daemon
- Slip
- Slirp
- Pcap

Vier dieser Typen erlauben den Pakettausch mit dem Host. Diese sind: TUN/TAP, Ethertap, Slip und Slirp. Entweder können diese direkt aus dem Internet erreicht werden, (über den Host) oder können untereinander völlig autonom kommunizieren.

Switch Daemon und Multicast

Diese zwei Typen ermöglichen ein komplett virtuelles Netzwerk, welches mit der physikalischen Schnittstelle nichts zu tun hat. Damit können virtuelle Rechner verbunden werden, ohne dass diese von aussen über das Netzwerkinterface erreichbar wären. Aus diesem Grund sind sie für das Hosting weniger geeignet, denn mit dem Switch Deamon wird lediglich ein weiterer Prozess eingesetzt, welcher gekillt werden kann.

Pcap

Der Pcap-Transport ist ein „read only" Modus. Damit können Pakete gelesen und gefiltert werden. Dieser Transporttyp eignet sich besonders gut für Sniffer oder Intrusion Detection Systeme (IDS). Für das Webhosting kommt es ebenfalls nicht in Frage.

Slirp

Slirp ermöglicht ein virtuelles Netzwerk ohne Administratorenrechte aufzusetzen. Da wir es nicht wünschen, dass jemand anders als der Hostadministrator im Neztwerkbereich etwas einstellen kann, werden wir auch diesen Typ für unsere Anwendung nicht berücksichtigen.

Slip

Auf der User Mode Linux Kernel Homepage wird von diesem Typ abgeraten, ausser man kann aus irgendwelchen Gründen weder Ethertap noch TUN/TAP einsetzen. Das Slip Protokoll kann im Gegensatz zu den beiden Erstgenannten, welche bereits auf Layer 2 mit Ethernetframes arbeiten, nur IP-Pakete austauschen.

Ethertap

Ethertap ist dann geeignet, wenn man direkt mit Host über Ethernetframes kommunizieren möchte und einen 2.2er Kernel einsetzt. Ethertap ist somit veraltet.

TUN/TAP

Wie mit Ethertap ist es auch mit TUN/TAP möglich, auf dem OSI Layer 2 so genannte Ethernetframes und darüber IP-Pakete mit dem Host auszutauschen. TUN/TAP ist auf den 2.4er Kernel zugeschnitten.
Wir arbeiten zwar mit dem 2.6er Kernel, aber es gibt allgemein noch fast keine Erfahrung mit UML und dem 2.6er Kernel. Wir haben keinen weiteren Typ für das virtuelle Netzwerk gefunden, der für den 2.6er Kernel optimiert wäre und somit verwendet werden könnte. TUN/TAP bietet auf einfache, flexible Weise die nötige Kommunikationsmöglichkeit zwischen UML, Host und der Aussenwelt.

Wie funktioniert TUN/TAP

TUN/TAP ist ein Treiber, welcher nichts anderes macht, als Pakete zu empfangen und weiterzuleiten. Häufig wird TUN/TAP als „Point to Point" – Protokoll oder als Netzwerkgerät bezeichnet. Diese Vorstellung ist zwar nicht ganz korrekt, hilft uns aber für deren Verständnis. Anstatt dass die Pakete von einem physikalischen Interface

kommen, werden Sie von einem so genannten „User Space Program“ empfangen und wieder an ein solches gesendet. Der gesamte Kommunikationsvorgang geschieht also nur virtuell.
TUN/TAP umfasst exakt dieselben Eigenschaften, wie wenn das Netzwerk über physikalische Netzwerkschnittstellen aufgebaut würde. Der Grund ist, dass TUN mit IP-Paketen und TAP mit den Ethernet-Paketen arbeitet. Diese raffinierte Eigenschaft trägt dazu bei, dass jeder, der eine Ahnung von Netzwerken hat, mit diesen Netzwerkprotokollen zurechtkommt.[6]
Entwickelt wurde TUN/TAP ursprünglich nicht für User Mode Linux, sondern für Einsatz mit VPNs.[7] Von der oben beschriebenen Funktionalität eignet sich TUN/TAP ausgezeichnet für die Vernetzung von UML-Instanzen.

Eine kurze Einarbeitungszeit braucht es aber schon um zu verstehen, wie das ganze nun genau funktioniert. Auf dem Host wird ein „Tap-Device“ in die Liste der Netzwerkschnittstellen aufgenommen. Die IP-Adresse desselben muss in den Bootoptionen der jeweiligen UML-Instanz mitgegeben werden. (Siehe Print Screen „`ifconfig -a`“)

[6] http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt

[7] http://vtun.sourceforge.net

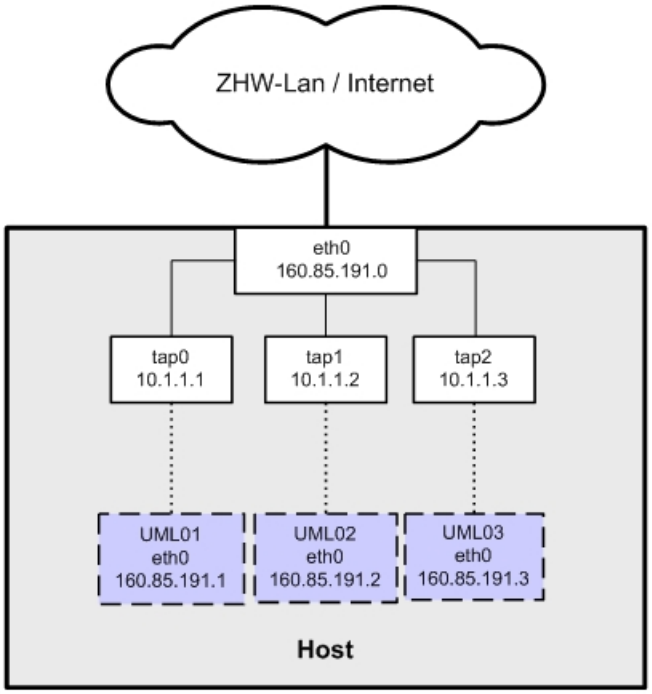
```
eth0      Link encap:Ethernet  HWaddr 00:04:75:C1:0C:00
          inet addr:160.85.191.0  Bcast:160.85.191.255  Mask:255.255.224.0
          inet6 addr: fe80::204:75ff:fec1:c00/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8704609 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3188 errors:0 dropped:0 overruns:0 carrier:1
          collisions:173 txqueuelen:1000
          RX bytes:1013965672 (966.9 MiB)  TX bytes:369035 (360.3 KiB)
          Interrupt:9 Base address:0xd400

tap0      Link encap:Ethernet  HWaddr 00:FF:8F:56:D4:30
          inet addr:10.1.1.1  Bcast:10.255.255.255  Mask:255.255.255.255
          inet6 addr: fe80::2ff:8fff:fe56:d430/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21 errors:0 dropped:0 overruns:0 frame:0
          TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1600 (1.5 KiB)  TX bytes:3230 (3.1 KiB)

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
160.85.191.4  *               255.255.255.255 UH    0      0        0 tap0
160.85.160.0    *               255.255.224.0   U     0      0        0 eth0
default         edugw.zhwin.ch  0.0.0.0         UG    0      0        0 eth0
```

Netzwerkkonfiguration des Hosts

Die UML-Instanz braucht aber auch noch eine IP-Adresse mit welcher sie angesprochen werden kann. Der Host fungiert dabei als Router. Dies bedeutet, dass auf dem Hostrechner IP-Forewarding eingeschaltet werden muss. Für jede UML-Instanz wird ein eigenes TAP-Device auf dem Host eingerichtet. Dies muss mit Rootrechten geschehen, allerdings besteht danach die Möglichkeit, dass die konfigurierte Schnittstelle von einem Userprogramm (UML-Kernel) genutzt wird. Damit wird nun auch klar, weshalb TUN/TAP als Point-to-Point Treiber bezeichnet wird. Auf der untenstehenden Abbildung ist die Konfiguration mit drei UML-Instanzen graphisch dargestellt.



Netzwerkkonfiguration mit drei UMLs

Aus obiger Abbildung ist auch ersichtlich, dass mit dieser Konfiguration die einzelnen Instanzen nur über das Hostsystem miteinander kommunizieren können.

Die privaten IP-Adressen der tapX Interfaces auf der Abbildung haben einen speziellen Grund. Wenn man jedem tapX, was eigentlich logisch wäre, eine vom Internet ansprechbare IP-Adresse geben würde, bräuchte man pro UML-Instanz zwei öffentliche IP-Adressen. Weil diese Adressen knapp und heutzutage auch teuer sind, wäre es gut, wenn man dies anders lösen könnte. Den UML-Instanzen kann man keine privaten IP-Adressen zuweisen, weil diese danach nicht mehr ohne weiteres von aussen ansprechbar sind. Das tapX Device stellt im Grunde genommen nur eine Kommunikationsschnittstelle dar und muss nicht zwingend von aussen angesprochen werden können. Deshalb macht es Sinn, pro UML-Instanz nur eine teure IP-Adresse zu verwenden und dem tapX Device eine Adresse aus dem privaten Bereich zuzuweisen.

Obwohl diese Konfigurationsart etwas ungewohnt aussieht, funktioniert sie einwandfrei und bedeutet keinen grossen Konfigurationsaufwand.

Bridge als Alternative

Als Alternative kann das virtuelle Netzwerk mittels einer Bridge realisiert werden. Dafür muss der Hostkernels die Funktion „802.1d Ethernet Bridging“ unterstützen. Danach wird der eine TUN/TAP Endpunkt an diese Bridge angehängt und der Andere wie vorher am UML-Netzwerkdevice. Durch diese Methode lässt sich ein Ethernet in verschiedene Segmente einteilen, wodurch ARP-Broadcasts eingeschränkt werden. Für diese Arbeit sehen wir allerdings keine Notwendigkeit für die Anwendung einer Bridge.

5. Sicherheit

Einzelne Aspekte aus dem Thema Sicherheit sind in den vorangegangenen Kapiteln bereits kurz erwähnt oder angeschnitten worden. In diesem Kapitel sollen die wichtigsten Sicherheitsaspekte nochmals detailliert betrachtet sowie wichtige Zusammenhänge erläutert werden.

Die Möglichkeiten mit UML sind nahezu unbeschränkt. Hingegen muss jeder UML Benutzer wissen, was er macht und darf dabei das Thema Sicherheit nicht aus den Augen lassen. Vergleichsweise schnell wird ein tolles Feature zum Laufen gebracht, mit welchem gleichzeitig eine grosse Sicherheitslücke geöffnet werden kann.

5.1. Patchmanagement

Vorbemerkungen

Die Firma Netzschmiede möchte im Rahmen dieser Projektarbeit erfahren ob und wie koordiniert Security-Patches und Updates für UML-Hosting eingespielt werden könnten. Dabei gilt es allerdings einige Punkte zu bedenken

- auf einer UML-Instanz wird auch hauptsächlich als „root" Software installiert.
- Auf einem Host können verschiedene Distributionen als UML laufen, welche unterschiedliche Patches erfordern.
- Möchte ein Kunde, der UML als Webserver nutzt, dass ihm der Hostadministrator auf seiner UML-Instanz Patches einspielt, mit der Gefahr, dass danach eigene Webprojekte nicht mehr richtig laufen?
- Ein Kunde, welcher in der Lage ist eine Linux Maschine als Webserver zu betreiben, sollte auch in der Lage sein, diese betriebssystemmässig auf dem aktuellen Stand zu halten.
- Eindeutig zur Aufgabe des Providers zählen wir das Patchen und kompilieren der Kernel (Host- und UML-Kernel).

Was spricht gegen ein Patchmanagement

Will man als Host-Administrator die virtuellen Rechner der Kunden patchen, muss man Rootrechte auf den entsprechenden UML-Instanzen haben. Herr Moser von der Firma Netzschmiede GmbH hat uns mitgeteilt, dass es nicht geplant sei verschiedene Distributionen als UML-Rechner anzubieten. Falls man diese trotzdem ausführen wolle, werden für jede Distribution eigene Patches benötigt, da die meisten untereinander nicht kompatibel sind. Dadurch entsteht ein grosser Mehraufwand für das Patchmanagement.
Selbst wenn alle Kunden mit derselben Distribution arbeiten, betreiben nicht alle dieselben Dienste und Versionen auf den UML-Instanzen (PHP, mySQL etc.).
Wir sind der Meinung, wenn ein Kunde UML nutzt, tut er dies hauptsächlich wegen dem vollen Rootaccess und der hohen Flexibilität für Dienste und Applikationen. Speziell durch die hohe Flexibilität wird dem Kunden ermöglicht, eigene individuelle Lösungen zu betreiben. Werden nun durch eine zentrale Stelle (Hostadministrator) auch innerhalb der UML-Instanzen wild Patches eingespielt, besteht die Gefahr, dass eigene Projekte nicht mehr wunschgemäss laufen. Unserer Meinung nach, ist es für den Hostadministrator fast unmöglich genau zu wissen was, wo und wie auf all den verschiedenen UML-Instanzen läuft. Hingegen genau dieses Wissen wäre von entscheidender Bedeutung für ein zentrales Patchmanagement, sei es auch nur um die Patches zentral zur Verfügung zu stellen. Hingegen darf davon ausgegangen werden, dass ein UML-Kunde gewisse Linuxgrundkenntnisse mitbringt, ja sogar besitzen muss, um seinen eigenen Server zu betreiben.
Auf Grund dieser Tatsachen soll den Kunden die Verantwortung für periodische Security- und Systemupdates übertragen werden. Der UML-Provider soll hingegen die Einhaltung einer solchen Regelung überprüfen und dem Kunden auf Wunsch Unterstützung für das Patchmanagement bieten. Speziell unter Debian kann ein UML-Stystem einfach mit dem Befehl „`apt-get upgrade`“ auf dem aktuellen Stand gehalten werden.

Was kann der Provider unternehmen?

Nebst einem tadellosen Support muss der UML-Provider die angebotenen Filesysteme, besonders für Neukunden, auf dem aktuellen Stand halten. Wird ein eigenes Filesystem

für jede UML-Instanz verwendet, kann dies relativ einfach mit einem Templatefilesystem, welches mit wenig Aufwand ständig auf dem aktuellen Stand gehalten werden kann, realisiert werden. Wie bereits im Abschnitt COW-Files erwähnt wurde, hat ein Aktualisieren des Backingfiles zwingend neue COW-Files zur Folge, was ein solches Konzept wiederum fast verunmöglicht.
Eine weitere Möglichkeit wäre natürlich, dass der Provider periodisch allen seinen Kunden ein neues, aktuelles Filesystem zur Verfügung stellt. Dabei müsste allerdings der Kunde nach jedem Update seine Daten und Projekte auf das neue Filesystem migrieren. Dieses Vorgehen würde von den Kunden wohl kaum längerfristig akzeptiert werden.

Mounten eines UML-Filesystems vom Hostadministrator

Eines sollte man sich bewusst sein, auch wenn der Hostsadministrator das Root-Passwort der einzelnen UML-Instanzen nicht kennt, besteht für ihn prinzipiell die Möglichkeit die Daten der einzelnen Filesysteme anzuschauen, indem er das Filesystem mountet. In unserem Test lies sich allerdings keine COW-Datei mounten. Um die Daten auch noch lesen zu können, dürfen diese natürlich nicht durch den Benutzer verschlüsselt worden sein.

Fazit

Wir denken nicht, dass es von den Kunden wirklich gefordert wird, dass die Hostingfirma das Einspielen von Security Patches übernehmen soll. Auch für die Hostingfirma stellt es einen grossen Aufwand dar, welcher teilweise eingespart werden könnte, indem dieser auf den Kunden übertragen wird.
Wir sind der Meinung, dass ein Kunde in der Lage sein sollte den eigenen UML-Rechner selbständig zu betreiben, ansonsten ist es besser, wenn er auf die konventionellen Hostingangebote zurückgreift.

5.2. Backupkonzept für UML-Instanzen

Einleitend sei erwähnt, dass es viele verschiedene Backup Strategien gibt. Wir wollen hier eine Möglichkeit aufzeigen, die wir als sinnvoll erachten. Des Weiteren beschäftigen wir

uns nur mit dem Backup für die UML-Instanzen und deren Daten in Bezug auf die Frage wie gesichert werden soll. Wir gehen nicht darauf ein, wie oft und auf welche Medien die Sicherungen durchgeführt werden sollen, weil das erstens von der Infrastruktur des Hostingproviders und zweitens von der Wichtigkeit der Daten abhängt, welche von jedem Kunden individuell beurteilt werden müssen.

Bei dem Backup im Zusammenhang mit UML, sollten zwei Dinge unbedingt getrennt, unabhängig betrachtet und beurteilt werden.

- Daten (Website, Datenbank, etc.)
- UML- Instanz (Kernel und Filesystem mit Programmen, Tools, Patches etc.)

Bei einem Webhosting mit virtuellen UML-Servern ist es aber unter Umständen nicht ganz einfach diese zwei Dinge, die meistens eine differenzierte Backuppriorität aufweisen, sinnvoll zu trennen.

Daten

Verantwortlich für das Backup ist im Grunde genommen der Betreiber des physischen Servers. Dieser sollte allerdings nicht direkt auf die UML-Instanzen zugreifen können (nur so, könnte er einzelne Daten trennen).
Es wäre interessant das Backup der UML-Rechner auf dem Host machen zu können. Das wäre möglich durch das mounten des Hostfilesystem. Wie wir aber im Kapitel Hostfileaccess beschrieben haben, bringt dies erhebliche Sicherheitsprobleme mit sich. Eine andere Möglichkeit für ein einfaches Backup, wäre das Mounten eines physikalischen Laufwerks von einer UML-Instanz aus.
Wir haben dies ausprobiert und auch nach mehrmaligen Versuchen war es nicht möglich ein physikalisches Laufwerk von der UML-Instanz aus zu mounten.

Unseres Erachtens besteht die sinnvollste Möglichkeit darin, den Kunden der einen „eigenen“ Server hat, auch in die Backupverantwortung einzubeziehen. Ihm könnte beispielsweise auf einem Fileserver ein Share zur Verfügung gestellt werden, auf welcher

er seine Daten selber sichern kann (z.B. Datenbankfiles). Mögliche Lösungen wären hierfür Protokolle wie SFTP, Samba oder NFS. Leider haben in der Vergangenheit die Protokolle Samba und NFS mehrere Sicherheitslücken aufgewiesen, weshalb wir von deren Einsatz in einer UML Umgebung eher abraten.

Als einfache und sichere Backuplösung empfehlen wir die Daten mittels SFTP auf einen anderen physikalischen Server zu sichern. Bei SFTP handelt es sich um eine Erweiterung des FTP Protokolls, bei welchem die Daten bei der Übertragung verschlüsselt werden. Nicht zu vergessen ist, auch wenn der Datenverkehr nur Providerintern abläuft, dieser nicht gegen Sniffing geschützt ist. Alle Kunden haben auf ihrem Server Rootrechte und damit fast beliebige Möglichkeiten in diesem Netzwerk.

UML-Instanz

Von einer gesamten UML-Instanz (Kernel und Filesystem) kann auf zwei verschiedene Arten ein Backup erstellt werden. Bei der ersten Variante muss die UML-Instanz heruntergefahren werden. Danach können die Files von Kernel und Filesystem an einen sicheren Ort kopiert und bei Bedarf noch zur Archivierung komprimiert werden. Bei der zweiten Variante kann die UML-Instanz im laufenden Betrieb gesichert werden, was speziell für UML-Server eine sehr interessante Alternative ist. Zu diesem Zweck wird diese in einen „schlafenden" Zustand gesetzt. Über die „`uml_mconsole`" wird der Befehl „`sysrq s`" abgesetzt. Dies bewirkt, dass der UML-Kernel zuerst alle Daten zurück ins Filesystem schreibt, was sehr wichtig für ein konsistent bleibendes Filesystem ist. Auch danach wird wie in der ersten Variante das Filesystem wegkopiert und der UML-Kernel wieder in den normalen Zustand zurückversetzt. In diesem „schlafenden" Zustand ist leider der gesamte Server für kurze Zeit nicht verfügbar. Diese Onlinebackupprozedur haben wir in unserem Skript implementiert.

Wir sind der Meinung, dass ein solcher Systemsnapshot die herkömmliche Backupvariante, zum Beispiel mit SFTP, für Kundendaten nicht ersetzt. Hingegen könnte diese Funktion als zusätzliche Dienstleistung vom Hostingprovider angeboten werden.

5.3. Rechtliche Aspekte

In diesem Abschnitt möchten wir noch einige Gedanken zum Thema Recht und den daraus entstehenden Problemen aufgreifen.

Es kann durchaus sein, dass ein Kunde das Patchmanagement vernachlässigt, wenn es ihm übertragen wird. Dadurch besteht die Gefahr, dass es einem Angreifer gelingt eine Sicherheitslücke auszunutzen, um die UML-Instanz zu kompromittieren und wie bei einem realen Rechner weitere Attacken, zum Beispiel in das Netzwerk der Hostingfirma, fahren kann. Ein solcher Angriff kann sich noch schlimmer auswirken wenn bei einem Dritten wirklich Schaden angerichtet wird. Wird nun ein solcher Angriff zurückverfolgt, kommt man schnell auf den UML-Hostingprovider, denn schliesslich stammt ja diese IP-Adresse auch aus seinem Bereich. Aus diesem Grund erachten wir es als sehr wichtig, dass sich der Provider vertraglich so absichert, dass er nicht für Schäden, die einer seiner Kunden anrichtet oder für Datenverlust innerhalb einer UML-Instanz haftbar gemacht werden kann. Der Provider wird – um sich selbst zu schützen - bei Feststellung strafrechtlicher Vergehen seiner Kunden, den Service für den fehlbaren Kunden umgehend einstellen, sowie Strafanzeige einreichen müssen. Um die Restrisiken möglichst einzuschränken, soll das ausgearbeitete Vertragswerk durch einen Rechtsexperten begutachtet werden.

5.4. Zugriffe

In der folgenden Tabelle werden die verschieden Zugriffsmöglichkeiten zwischen Host und UML aufgezeigt.

Übersicht:

Zugriff vom Hostsystem auf UML (Konsole)	Ja
Zugriff von UML auf Hostsystem (Konsole)	Nein
Zugriff von UML auf andere UML (Konsole)	Nein
Zugriff von UML auf Hostsystem (SSH)	Ja
Zugriff vom Hostsystem auf UML (SSH)	Ja
Zugriff von UML auf andere UML (SSH)	Ja

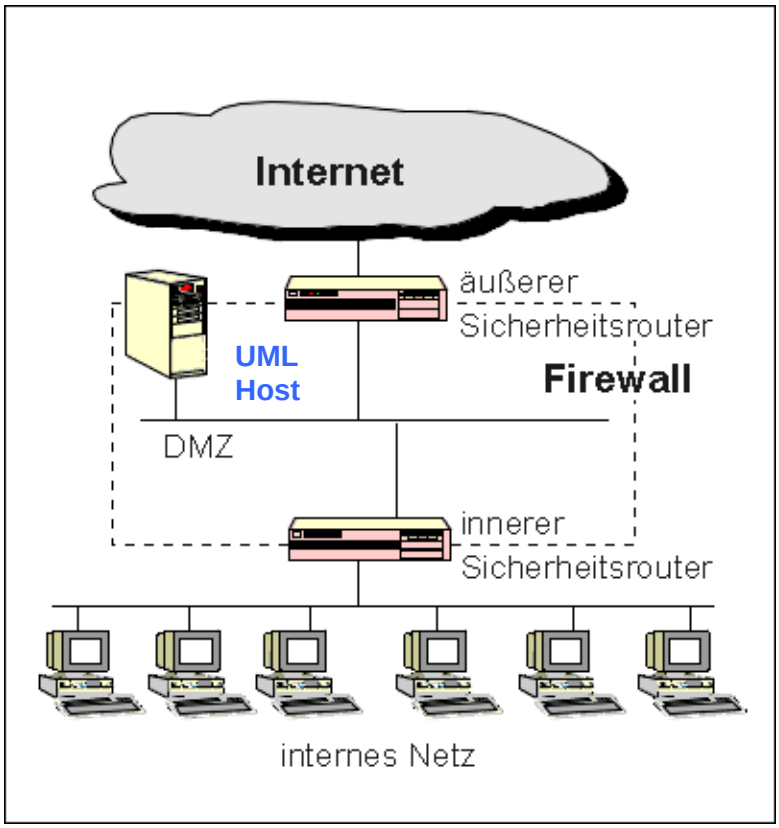
Zugriff von Hostsystem auf das Filesystem der UML	Ja, mounten (in laufendem Zustande wird Filesystem zerstört)
Zugriff von UML auf das Filesystem des Hostsystem	Je nach Einstellung (siehe „Host File Access“) nicht empfohlen
Zugriff von UML auf das Filesystem einer anderen UML	Nein
Zugriff von UML auf eine Festplatte des Hosts (mounten)	Nein

Wie schon unter Host File Access erwähnt, sollte unbedingt darauf geachtet werden, dass die Option „Host Filesystem“ bei dem UML-Kernel deaktiviert ist. Damit können keine UML-Instanzen auf das Hostsystem zugreifen.

5.5. Standort Hostserver

Mit dem Standort des Hostservers ist nicht der physische, sondern der logische Standort gemeint.

Bei unserer Empfehlung, das Patchmanagement dem Kunden zu überlassen, besteht die Gefahr, dass Patches nicht oder zu spät eingespielt werden. Da die UML-Nutzer noch Rootrechte auf ihrer Instanz verfügen, darf der Hostserver auf keinen Fall im Firmennetzwerk des Providers stehen. Es ist unbedingt notwendig, eine so genannte Demilitarisierte Zone (DMZ) einzurichten, bei welcher der Server, wie auf dem folgenden Bild, zwischen zwei Firewalls steht. Somit wird das eigene Firmennetzwerk effizient gegen Angriffe geschützt.



UML in DMZ [8]

5.6. Zuverlässigkeit und Verfügbarkeit

Ein wichtiger Bestandteil des Datensicherungskonzeptes soll das zur Verfügung stellen eines zusätzlichen Fileservers sein. Dies bedeutet das einerseits Mehrkosten durch die zusätzliche Hardware entstehen, andererseits sich auch der Wartungs- und Administrationsaufwand für beide Server vergrössert. Die Mehrkosten, um beide Systeme zuverlässig und allenfalls sogar redundant zu betreiben, sind relativ hoch. Beim professionellen UML-Hosting sollte man eine hohe Verfügbarkeit garantieren können.

Da wir beide keine grosse Erfahrung im Clustern von Servern und hochverfügbaren Systemen, insbesondere unter Linux haben, möchten wir zu diesem Thema nur einige Grundsätze erwähnen. Es wäre unserer Meinung noch sinnvoll zu prüfen, ob und wie UML-

[8] http://amor.rz.hu-berlin.de/~h0271cbj/papers/rzm13/alex3.gif

Instanzen geclustert werden könnten, wenn deren Filesysteme auf einer gemeinsam SAN (Storage Area Network) liegt. Wie nun die Firma Netzschmiede die Hardware schlussendlich zusammenstellt, wird nicht zuletzt abhängig von den konkreten Kundenbedürfnissen sein, welche wir nicht kennen. Als wichtig erachten wir, dass sich der Provider darüber Gedanken macht und sich dieser Problematik bewusst ist.
Das Mindeste in diesem Bereich ist das Betreiben von redundanten Festplatten (sprich RAID). Die restliche Hardware sollte wenn nicht schon redundant vorhanden, wenigstens innerhalb kürzester Zeit als Ersatzteile beschafft werden können.
Grundsätzlich stellen wir fest: Es gelten dieselben Regeln wie für das konventionelle Webhosting.

5.7. Starten von UML-Instanzen

Im Verlaufe der Arbeit stellte sich für uns die Frage, wer dann überhaupt eine UML Instanz starten darf. Einerseits wäre es ja praktisch, wenn dies der Kunde seinen Bedürfnissen entsprechend selber machen könnte. Andererseits braucht dieser für das Starten einen gültigen Benutzeraccount auf dem Hostrechner und muss sich darauf mit SSH einloggen können.
Mit etwas Aufwand besteht die Möglichkeit, dass ein User auf dem Host seine UML-Instanz starten kann. Der Hostadministrator muss dazu das TAP-Device (siehe: Wie funktioniert TUN/TAP) auf dem Host konfigurieren und den jeweiligen User die Berechtigung zur Benützung erteilen. Für diese Konfiguration sind zwingend Rootrechte erforderlich.

Vorteile

- Der Host Administrator wir etwas entlastet.
- Der UML Administrator ist nicht auf die Präsenzzeiten des Host Administrator angewiesen.
- Der UML Administrator kann die Instanz bei versehentlichem Herunterfahren wieder selbständig starten.

- Die UML-Instanz läuft dadurch automatisch im Kontext des jeweiligen Benutzers. Dadurch können individuellere Berechtigungen erteilt werden sowie viel detaillierte Aussagen bezüglich verursachten Traffics und Inanspruchnahme der CPU. Kurz das gesamte Reporting wird vereinfacht.

Nachteile

- Die Kunden erhalten durch den Benutzeraccount direkten Zugriff auf das Hostsystem.
- Der User könnte somit eigene UML-Kernel kompilieren oder eigene Filesysteme kreieren und auf dem Host laufen lassen.
- Der Benutzer kann beispielsweise Kernelsicherheitslücken wie jene vom 11. Juni 2004[9] ausnutzen und damit den gesamten Host einfrieren. (Voraussetzung: x86 System und Kernel <2.6.7)

Fazit

Unserer Meinung nach überwiegen die Nachteile ganz klar. Der Kunde kann als Hostbenutzer nicht mehr Dinge erledigen, welche er nicht auch in seiner UML-Box machen könnte. Die Entlastung des Host Administrators wird ebenfalls nicht von grossem Ausmass sein. Natürlich sieht das Ganze etwas anders aus, wenn der Provider seine Kunden kennt und ihnen vertrauen kann. Wir glauben zwar nicht, dass es die Absicht eines Kunden wäre, möglichst viele solcher Lücken auszunutzen oder möglichst oft seine UML-Instanz herunterzufahren, nur damit sie der Hostadministrator wieder hochfahren muss. Ausserdem besitzt der UML-Administrator nach wie vor die Möglichkeit mit dem Befehl

```
UML# shutdown –r now      (reboot)
```

seine Instanz neuzustarten, wenn diese einmal läuft.

[9] http://linuxreviews.org/news/2004-06-11_kernel_crash

Unserer Meinung nach soll nur der Hostadministrator UML-Instanzen aufsetzen und starten können. Es macht hingegen aus Sicherheitsgründen Sinn, wenn die UML-Instanzen unter einem Benutzer mit eingeschränkter Berechtigung gestartet werden.

Für erhöhte Sicherheitsansprüche bietet Linux die Möglichkeit, trotz Verwendung des SKAS-Modus, die UML-Instanzen jeweils in einer eigenen chroot-Umgebung oder Jail laufen zu lassen. Dadurch wird nochmals eine zusätzliche „Barriere" zwischen Host und UML geschaffen. Diese Technik konnte leider aus zeitlichen Gründen nicht mehr im Rahmen dieser PA berücksichtigt werden. In der neusten Version von UMLazi[10] ist dies nach Aussagen des Entwicklers integriert (siehe: Verwaltungstools für UML).
Netzwerkseitig wäre es dem Hostadministrator möglich, Einschränkungen z.B. mit Firewalls zu schaffen. Allerdings verliert dadurch das Angebot des eigenen Servers an Attraktivität.

5.8. SSH-Schlüsselmanagement

Der SSH Dienst ist äusserst praktisch und wird heute für vieles gebraucht. Um auf eine UML-Instanz zuzugreifen, ist SSH eine praktische und sichere Technik.
Ganz wichtig ist aber, dass der UML-Provider bei der Konfiguration seines Systems keine Fehler macht. Hat man es erst einmal geschafft, ein UML-Filesystem zu kreieren, ist die Verlockung gross, dieses zu kopieren, damit mehrere Instanzen gestartet werden können. Aber aufgepasst: Jedes kopierte Filesystem (auch COW) hat nun das gleiche Schlüsselpaar. Für den öffentlichen Schlüssel wäre das ja egal. Aber der private Schlüssel des Hostkeys muss zwingend einmalig sein.
Ansonsten kann theoretisch jeder Administrator einer UML-Instanz den verschlüsselten Verkehr einer anderen Instanz abhören.
Es ist folglich ausserordentlich wichtig für jedes Filesystem, sofern diese nicht einzeln erstellt sondern kopiert werden, ein neues Schlüsselpaar zu erzeugen.

[10] http://umlazi.org/2004/06/26/jail-security

Dies kann beispielsweise mit dem Befehl „ssh-keygen“ geschehen.

```
#/usr/bin/ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N ''
```

Nach Eingabe des Befehls geschieht folgender Ablauf und der neue Hostkey wird generiert:

```
Generating public/private rsa key pair.
/etc/ssh/ssh_host_rsa_key already exists.
Overwrite (y/n)? y
Your identification has been saved in
/etc/ssh/ssh_host_rsa_key.
Your public key has been saved in
/etc/ssh/ssh_host_rsa_key.pub.
The key fingerprint is:
02:b1:72:3b:59:26:84:12:66:69:6c:ca:fb:ac:1d:e6 root@uml03
```

Alternativ kann der Schalter –N '' weggelassen werden, um auch eine neue Passphrase zu erzeugen.

6. Verwaltungstools für UML

Für die Verwaltung von UML-Instanzen stehen verschiedene Projekte zur Auswahl, alle mit demselben Ziel, die Administration virtueller Rechner zu vereinfachen. Wir haben im Rahmen unserer Projektarbeit einige Tools getestet, insbesondere auf ihre Tauglichkeit in einer produktiven Umgebung.

6.1. uml_utilities

Die uml_utilities werden von den Entwicklern aus dem UML-Kernelprojekt angeboten und stehen somit auf der Kernelhomepage in der Version uml_utilities_20040406.tar.bz2[11] zum Download zur Verfügung. Es handelt sich dabei um folgende Hilfsprogramme:

- port-helper — Wird von Konsolen benötigt, welche sich auf Xterms oder Ports verbinden
- tunctl — Konfigurationstool um Tap-Devices zu erstellen und zu löschen
- uml_net — Setuidbinary um automatisch Tap-Devices zu konfigurieren
- uml_switch — Userprogramm, welches ein virtueller Switch/Hub simuliert. Es wird für Deamonnetzwerke benötigt.
- uml_mconsole — Verwaltungskonsole für UML-Instanzen über die folgende Befehle können abgesetzt werden: version, halt, reboot, config, remove, sysrq, help, cad, stopp und go.
- uml_mkcow — Erstellt zu einem Filesystem eine dazugehörige COW-Datei.
- uml_moo — Damit kann der Inhalt einer COW-Datei zurück in das Backingfilesystem gespeichert werden.

6.2. umlmgr

Beim umlmgr[12] handelt es sich um ein Curses basiertes Userinterface, welches die gestarteten UML-Instanzen übersichtlich darstellt und über welches verschiedene mconsolen-Befehle abgesetzt werden können. Der umlmgr steht in der Version 0.9.0 bereit und benötigt zur Ausführung Python 2.2 oder höher. Die Installation erfolgte ohne

[11] http://prdownloads.sourceforge.net/user-mode-linux/uml_utilities_20040406.tar.bz2

[12] http://www.rhythm.cx/~steve/devel/umlmgr

Probleme, jedoch konnten keine 2.6er UML-Kernel in der aktuellen Version verwaltet werden. Die 2.6er UML-Instanzen wurden abrupt durch den Start von umlmgr abgeschossen. Mit dem Debianeigenen 2.4.24 Gastkernel liessen sich alle Systeminformationen der Instanzen übersichtlich anzeigen und die Funktionen wie start, stopp und reboot liessen sich ebenfalls problemlos ausführen.

6.3. UMLazi

UMLazi[13] ist im Vergleich zum umlmgr ein etwas umfangreicheres Projekt und bietet insbesondere die Möglichkeiten aufgrund von Templates neue UML-Instanzen zu konfigurieren, aufzusetzen und zu verwalten. Eine Installationsanleitung, so wie spärliche Dokumentation ist auf der Projektseite vorhanden. In der von uns getesteten Version 1.0.3.2 funktionierte leider das Projekt mit dem eigenen 2.6er Kernel und Filesystem nicht. Unabhängig von der Templatekonfiguration wurde zudem ohne Warnung die gesamte Netzwerkeinstellung des Hostsystems automatisch umkonfiguriert, wodurch unser Setup nicht mehr ordnungsgemäss funktionierte. In der Zwischenzeit wurde eine neue Version 1.0.5.1 herausgegeben. Aber auch dort ist noch von Problemen mit 2.6er Kerneln die Rede. Aus zeitlichen Gründen konnten wir leider diese neueste Version nicht mehr testen.

6.4. UMLd

UMLd[14] ist ein Projekt, welches von David Coulson ins Leben gerufen wurde und auf PHP und MySQL basiert. Der Daemon stellt eine Schnittstelle zu einem TCP-Socket zur Verfügung und erlaubt es, einfache Clients zu beschreiben. Die einzelnen UML-Instanzen können so von einem Benutzer oder Administrator verwaltet werden. Weil überhaupt keine Dokumentation vorhanden ist, haben wir leider das Projekt trotz grossem Zeitaufwand nicht zum Laufen gebracht.

[13] http://umlazi.org

[14] http://uml.openconsultancy.com/umld

6.5. Vmon

vmon[15] ist ein kompaktes Werkzeug für das Handhaben und die Überwachung von UML-Instanzen. Der Funktionsumfang beschränkt sich lediglich auf starten, überwachen und stoppen von UML-Instanzen sowie die Konfiguration über ini-Dateien. Ursprünglich wurde das Tool geschrieben, um etwa zehn virtuelle Maschinen, welche auf zwei physikalische Rechner verteilt sind, zu administrieren. Der Programmierer zeigt sich einerseits stolz darauf vmon der Öffentlichkeit zu präsentieren und wie er schreibt, halb-stolz darauf, keine Unterlagen für das Installieren und Verwenden dieser Software zur Verfügung zu stellen[16]. Aufgrund dieser tollen Voraussetzungen konnte leider auch dieses in Python geschriebene Projekt nicht in Betrieb genommen werden.

6.6. VNUML

VNUML[17] (Virtual Network User Mode Linux) ist ein Virtualisierungswerkzeug, welches entworfen wurde, um schnell komplexe Netzwerksimulationen zu definieren und mit dem Einsatz von User Mode Linux durchzuführen. Entwickelt wurde VNUML im Rahmen des Euro6IX-Forschungsprojektes an der Technischen Universität Madrid um IPv6-Szenarien unter Linux zu simulieren. Mittlerweile hat sich VNUML zu einem nützlichen Werkzeug etabliert, um Linuxbasierte Netzwerkszenarien zu simulieren. VNUML beabsichtigt beliebig komplexe Testfälle für Netzwerkapplikationen und Dienste über mehrere virtuelle Knoten innerhalb eines Linuxrechners durchzuführen. Das Ganze kann nun ohne grosse Investition und Administration erfolgen. VNUML besteht aus den beiden Komponenten VNUML-Sprache, welche die Simulationen in XML beschreibt, sowie einem eigenen Parser, welcher die Simulation durchführt, verwaltet und alle komplizierten Details von UML gegenüber dem Benutzer versteckt.

[15] http://goldenspud.com/vmon

[16] http://goldenspud.com/webrog/archives/2004/02/10/vmon-unleashed

[17] http://jungla.dit.upm.es/~vnuml

Aufgrund des Einsatzgebietes von VNUML sind wir im Rahmen dieser Projektarbeit nicht mehr weiter auf dieses Projekt eingegangen. Dem interessierten Leser steht hingegen auf der Projektseite sehr umfangreiches Material wie Tutorial, Installationsanleitung, Benutzerhandbuch, VNUML-Sprachreferenz sowie eine FAQ-Seite zur Verfügung.

7. HowTo

7.1. Hostkernel kompilieren

Um erfolgreich einen 2.6.3 Hostkernel zu kompilieren müssen folgende Pakete installiert sein:

libncurses-dev

insmod

module-init-tools

libqt-dev (für grafische Kernelkonfiguration unter KDE)

Als erstes müssen die Kernelsourcen[18] heruntergeladen und ausgepackt werden. Danach muss der passende SKAS-Patch[19] angewendet werden.

```
HOST# cd /usr/src
HOST# tar –xjvf linux-2.6.3.tar.bz2
HOST# ln –s linux-2.6.3 linux
HOST# cd linux
HOST# patch –p1 <../host-skas3-2.6.3-v1.patch
HOST# make menuconfig
```

Unter “Processor type and features” soll /proc/mm ausgewählt werden und TUN/TAP unter „Network device support“ als Modul selektiert werden. Ansonsten kann der Hostkernel weiter nach belieben konfiguriert werden.

```
HOST# make
HOST# make modules
HOST# make modules_install
HOST# make install
```

[18] http://www.kernel.org

[19] http://user-mode-linux.sourceforge.net/dl-sf.html oder inoffiziell unter http://www.user-mode-linux.org/~blaisorblade

Der neue Kernel wird damit automatisch installiert und in den Bootloader eingetragen. Der vorsichtige Administrator vergewissert sich, dass er Notfalls noch mit entsprechendem Eintrag in der Bootloader.config seinen alten Kernel booten kann.

7.2. UML-Kernel kompilieren

Als Gastkernel möchten wir einen 2.6.4/7er Kernel kompilieren. Wichtig ist, dass dieser im Gegensatz zu einem Hostkernel nicht in /usr/src/linux kompiliert wird, sondern als eigene Architektur in einem beliebigen Verzeichnis. Die Kernelsource muss wiederum mit einem UML-Patch[20] gepatcht werden.

```
HOST# cd /usr/src
HOST# rm linux
HOST# tar –xjvf linux-2.6.4.tar.bz2
HOST# cd linux-2.6.4
HOST# bzcat ../uml-patch-2.6.4-1.bz2 | pacht –p1
HOST# pacht –p1 <../host+UML-skas3-2.6.3-v1.patch
HOST# make menuconfig ARCH=um
```

Wir kompilieren sicherheitshalber alle benötigten Optionen fest in den Kernel (kein Modulsupport). /proc/mm soll auch für das Gastsystem ausgewählt werden und hostfs deaktiviert werden.

```
HOST# make scripts
HOST# make scripts ARCH=um
HOST# make linux ARCH=um
```

Der kompilierte UML-Kernel steht nun im selben Verzeichnis als *linux* zur Verfügung und kann an beliebige Stelle zur Ausführung kopiert werden.

7.3. UML-Tools kompilieren und installieren

Um die UML-Tools kompilieren zu können muss im Voraus noch das Paket *libreadline-dev* installiert werden. Danach kann mit folgenden Befehlen fortgefahren werden.

[20] http://user-mode-linux.sourceforge.net/dl-sf.html oder inoffiziell unter http://www.user-mode-linux.org/~blaisorblade

```
HOST# tar –xjvf uml_utilities_20040406.tar.bz2
HOST# cd tools
HOST# make all
HOST# make install DESTDIR=”gewünschtes
Installationsverzeichnis”
```

Ohne Angabe eines Zielverzeichnisses werden die Tools standardmässig in /usr/bin/ installiert.

7.4. Eigenes Rootfilesystem mit Debian erstellen

Das Debianrootfilesystem wird mit dem Tool *debootstrap* aufgesetzt. Das Rootfilesystem wird in einer einzigen Datei erzeugt. Als Filesystemtyp wählen wir ReiserFS oder ein sonstiges Journalingfilesystem.

```
HOST# cd /uml
HOST# dd if=/dev/zero of=./root_fs-reiser bs=1024k count=500
HOST# mkreiserfs -f root_fs-reiser
HOST# losetup /dev/loop0 root_fs-reiser
HOST# mount /dev/loop0 /mnt
HOST# cd /mnt
HOST# debootstrap sarge . ftp://sunsite.cnlab-
switch.ch/mirror/debian
HOST# cp /etc/network/interfaces /mnt/etc/network/interfaces
HOST# chroot /mnt
```

Nun haben wir ein minimales Basissystem installiert und müssen noch einige Dinge manuell konfigurieren, damit auch eine UML-Instanz erfolgreich gestartet werden kann. Das Ganze wird in einer chroot-Umgebung erledigt.

```
HOST# apt-setup
HOST# cat /etc/apt/sources.list | sed -e 's/stable/sarge/'
>/etc/apt/sources.list
HOST# mount /proc
HOST# apt-get update && apt-get upgrade
HOST# dselect --expert
HOST# dpkg -P pcmcia-cs
HOST# dpkg-reconfigure locales
HOST# dpkg-reconfigure console-data
HOST# tzconfig
HOST# echo "umlRechnerName" >/etc/hostname
```

```
HOST# echo "127.0.0.1 localhost" >/etc/hosts
HOST# echo "160.85.191.1 umlRechnerName.zhwin.ch"
>>/etc/hosts
HOST# echo "search zhwin.ch" >/etc/resolv.conf
HOST# echo "nameserver 160.85.195.195" >>/etc/resolv.conf
HOST# nano etc/network/interfaces
HOST# apt-get install ssh tcpdump traceroute
```

Mit dem letzten Befehl können den eigenen Bedürfnissen entsprechend, selbstverständlich beliebig viele Pakete wie zum Beispiel *apache, php4* und *mysql-server* zur Installation mitgegeben werden. Grundsätzlich sind alle diese Pakete optional.

```
HOST# apt-get clean
```

Debian GNU/Linux (Sarge) unterstützt durch die GlibC die Native Posix Threads Library (NPTL). Weil diese Bibliothek durch den 2.6er UML-Kernel noch nicht unterstützt wird, muss das Verzeichnis /lib/tls entfernt oder umbenannt werden. TLS steht für Thread Local Storage. Ansonsten erhält man beim booten folgende Fehlermeldung „INIT: Id "1" respawning too fast: disabled for 5 minutes“.

```
HOST# mv /lib/tls /lib/tls.disable
```

Sofern man nicht wünscht, dass nach dem booten mit diesem Filesystem der gesamte Bildschirm mit virtuellen Konsolen geflutet wird, müssen noch in der /etc/inittab alle überflüssigen virtuellen Konsolen wie folgt auskommentiert und ergänzt werden und in der /etc/securetty die tty0 hinzugefügt werden, damit der Loginpromt in der selben Shell erscheint, von wo der Kernel gestartet wurde.

```
HOST# nano /etc/inittab
     0:2345:respawn:/sbin/getty 38400 tty0
     1:2345:respawn:/sbin/getty 38400 tty1
     #2:23:respawn:/sbin/getty 38400 tty2
     #3:23:respawn:/sbin/getty 38400 tty3
     #4:23:respawn:/sbin/getty 38400 tty4
     #5:23:respawn:/sbin/getty 38400 tty5
     #6:23:respawn:/sbin/getty 38400 tty6
```

```
HOST# nano /etc/securetty
     # Standard consoles
     tty0
```

Die Datei /etc/fstab muss für die UML-Disks angepasst werden.

```
HOST# echo "/dev/ubd0 / reiserfs defaults 0 0" >/etc/fstab
HOST# echo "proc /proc proc defaults 0 0" >>/etc/fstab
HOST# cd /dev
HOST# for i in 0 1 2 3 4 5 6 7; do mknod ubd$i b 98 $i; done
HOST# umount /proc
HOST# exit
```

Nach diesen Schritten kann die chroot-Umgebung wieder verlassen werden. Es müssen noch alle Prozesse, die auf unser Filesystem zugreifen, beendet werden.

```
HOST# lsof -a /mnt/
HOST# cd –
HOST# umount /mnt
HOST# losetup –d /dev/loop0
```

Anschliessend kann die UML-Instanz gebootet werden und über ssh oder die virtuelle Konsole eingeloggt werden.

```
HOST# ./deb-264uml-1 ubd0=root_fs-reiser
eth0=tuntap,,,10.1.1.4 umid=uml04
```

7.5. UML-Filesystem vergrössern

Um ein UML-Filesystem zu vergrössern darf keine UML-Instanz mehr darauf zugreifen. Danach soll von dem zu vergrössernden Filesystem und den entsprechenden COW-Dateien ein Backup (Kopie) erstellt werden.

```
HOST# e2fsck –f dateiname
HOST# dd if=/dev/zero of=dateiname bs=1 count=0
seek=neueFileGrösse
HOST# resize2fs –p dateiname           // resize_reiserfs
HOST# e2fsck –f dateiname
```

7.6. Backup einer laufenden UML-Instanz

Es ist möglich ein Onlinebackup oder besser gesagt einen Snapshot einer laufenden UML-Instanz zu erstellen. Dies geschieht indem man die UML-Instanz pausieren lässt, die Daten im Speicher flusht, das Filesystem an einen sicheren Ort kopiert und danach die UML-Instanz weiter rennen lässt.

```
HOST# uml_mconsole umid
(uml_mconsole) stop
(uml_mconsole) sysrq s
HOST# cp /pfad/uml-filesystem /pfad/sicherer-ort
(uml_mconsole) go
(uml_mconsole) quit
```

Der Befehl „`sysrq s`“ stellt sicher, dass das Filesystem sich in einem konsistenten Zustand befindet.

7.7. Verwaltung von UML-Instanzen

Da wir kein Projekt gefunden haben, mit welchem UMLs wirklich einfach verwaltet und aufgesetzt werden konnte, haben wir ein kleines Konfigurations- und Verwaltungsscript (Shellskript) geschrieben. Damit lässt sich ein UML-Rechner konfigurieren, unter dem Benutzer umladmin starten, rebooten und stoppen. Am Schluss haben wir noch eine Backupfunktion implementiert, um einen Snapshot einer laufenden Instanz zu erstellen.

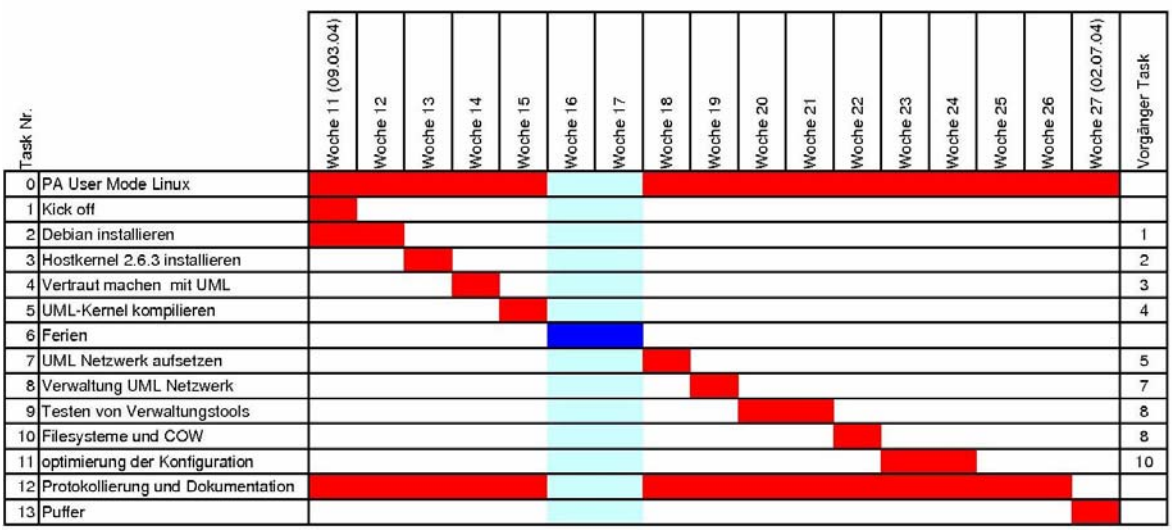
Ein Templateskript wird nach /etc/init.d/rechnerName kopiert und dort mit einem beliebigen Editor konfiguriert. Danach kann ein symbolischer Link nach /usr/bin erstellt werden, um das Skript ohne die lästigen Pfadangaben aufrufen zu können. Das Skript kann durch den Befehl „`update-rc.d umlSkriptName start 99 2 3 4 5 . stop 01 0 1 6 .`“ in den Startconfigs verlinkt werden, um die UMLs automatisch nach einem Hostreboot wieder zu starten.

Unser Skript setzt das Programm *screen* voraus. Dabei handelt es sich um ein nützliches Tool, um virtuelle Konsolen zu verwalten und sich später wieder auf eine bestehende Sitzung zu verbinden. Jeder Kernel wird in einer eigenen screen-Session gestartet und kann dort auch beobachtet werden. Von diesem Screen aus, kann man sich nun auch auf der UML-Instanz, quasi in der Ersatzkonsole, anmelden.

Es ist zu sagen, dass in diesem Skript - aus unserer Sicht - nur die wichtigsten Grundfunktionalitäten abgedeckt werden. Das Skript könnte bestimmt noch fast beliebig um weitere Funktionalität erweitert werden.

8. Projektablauf und Schlussfolgerungen

8.1. Projektplan



Projektplan

Der Projektplan war für uns ein wichtiges Instrument zur Steuerung dieser Arbeit. Wir hängten einen zwei A3 Seiten grossen Ausdruck an die Pinwand unseres Arbeitsplatzes auf. Anhand dieses Planes haben wir jeweils am Montag die Wochenziele festgelegt. Die Tagesziele konnten wir nicht immer erreichen. Diese waren nämlich bewusst sehr hoch angesetzt. Für die Grundinstallation des Debian Betriebssystems brauchten wir länger als in den entsprechenden Wochenzielen angestrebt.
Der Projektplan als Ganzes, hat sich als sehr realistisch erweisen. Es waren keine Korrekturen nötig.

8.2. Arbeitsaufeinteilung

Einige Arbeiten haben wir gemeinsam durchgeführt, damit meinen wir, dass keine fixe Zuteilung stattgefunden hat. Wir haben jeweils während dem Arbeiten aufgeteilt, wer was bis wann ausführt.

Wir haben aber schon zu Beginn definiert, wer schliesslich für welchen Teil verantwortlich ist.

Martin Gonzenbach: technischer Teil (Koordination der technischen Arbeiten, verantwortlich für die Hostmaschinen z.B. Backup)

Sandro Müller: Dokumentation und Kommunikation (protokollieren der technischen Arbeiten und Sitzungen, sowie verantwortlich für die Erstellung der Dokumentation)

Dabei ist es uns aber wichtig zu betonen, dass beide Studenten in gleichem Masse zum Gelingen der Arbeit beigetragen haben und dass die Einteilung vorgenommen wurde um Verantwortungsbereiche zu schaffen.

Diese Einteilung hat sich bestens bewährt. Aufgrund der grösseren Erfahrung von Martin Gonzenbach mit dem Linux-Betriebssystem machte es Sinn, dass er die technischen Arbeiten koordinierte.
Zu Beginn der Arbeit, waren wir beide hauptsächlich mit technischen Arbeiten beschäftigt.

8.3. Konsequentes Protokollieren

Von der ersten Stunde an, haben wir alle Arbeiten konsequent protokolliert. Wir waren der Meinung, dass es schwierig sein wird, von Anfang an den Text so zu gestalten, damit dieser direkt in die Dokumentation einfliessen kann. Diese Annahme stellte sich mit der Zeit als absolut richtig heraus, aber es hat sich bewährt, auf die Protokollierung zu setzen.
Wir verfassten diese Protokolle meist nur stichwortartig mit einigen Tabellen und Bildern. Sie dienten uns zur Erhaltung des Überblickes auf hervorragende Weise. Während des Schreibens der Dokumentation sind häufig Fragen aufgetaucht, welche wir mit den entsprechenden Protokollen mühelos klären konnten.

8.4. Zusammenarbeit mit der Partnerfirma und dem Dozenten

Netzschmiede GmbH

Unsere Kontaktperson bei der Firma Netzschmiede war Herr Simon Moser.
Die Zusammenarbeit mit Herrn Moser hat hervorragend geklappt. Herr Moser hat uns selbständig arbeiten lassen. Trotzdem hat er die notwendigen Inputs gegeben und ein grosses Interesse am Fortschritt der Arbeit gezeigt.
Regelmässig haben wir kleine Sitzungen durchgeführt und über den jeweiligen Stand der Arbeit informiert. Herr Moser hat uns auch seine Hilfe angeboten, falls wir irgendwo stecken bleiben sollten. Es war für uns sehr angenehm diese Rückendeckung zu spüren.

Herr Prof. Dr. A. Steffen

Herr Prof. Dr. Steffen hat uns auch grosse Freiheit in der Gestaltung und Ausführung der Projektarbeit gelassen. Dieses Vertrauen, das Herr Prof. Dr. Steffen in uns setzte, schätzten wir sehr. Jederzeit wenn wir Fragen hatten, durften wir uns bei Ihm melden und bekamen immer innert kürzester Zeit eine kompetente Antwort. Auch bei den regelmässigen Meetings zeigte Herr Prof. Dr. Steffen ein grosses Engagement für unsere Arbeit und für uns.
Wir danken Herrn Prof. Dr. Steffen für die gute Zusammenarbeit.

8.5. Schlusswort

Im Rahmen dieser Projektarbeit haben wir einen sehr tiefen Einblick in das UML-Projekt sowie die Linuxwelt besonders unter Debian erhalten. Die Debiandistribution war für uns beide komplettes Neuland und ist in keiner Art und Weise mit Suse vergleichbar. Dies hat sich besonders in der Anfangsphase mit Problemen beim Aufsetzen und Konfigurieren des Systems gezeigt. Der zweite, wesentliche Knackpunkt stellte danach der neue 2.6er Kernel dar, sei es für Host und UML. Während der Arbeit, konnten wir eine kontinuierliche Verbesserung der UML-Patches für den 2.6er Kernel feststellen.
Im Verlauf dieser Projektarbeit stellten wir fest, dass mit UML die verrücktesten Konfigurationen und Setups möglich sind, sei dies in den Bereichen Netzwerk oder

Kernelfunktionalität. Nicht zuletzt aus diesem Grund verbrachten wir viel Zeit mit probieren.

Besonders in den Bereichen Verwaltungs- und Konfigurationstools sehen wir noch wesentliches Verbesserungspotential. Es sind zwar Tools mit interessanten Ansätzen vorhanden, wobei aber keines für den 2.6er Kernel brauchbar ist. Gerne hätten wir auch noch die Möglichkeiten der UML-Hochverfügbarkeit betrachtet. Dieser Aspekt wäre besonders für den Einsatz in Enterprisecomputing und Serverkonsolidierung interessant. Mit den bestehenden Möglichkeiten betrachten wir UML als geeignet, um einen qualitativ hoch stehenden Hostingdienst den Kunden anzubieten.

Die Arbeiten mit UML waren wirklich lehrreich und haben zudem auch Spass gemacht.

9. Anhang

9.1. Shellskript zur einfachen Verwaltung von UMLs

```
#!/bin/sh
#
# Diese Skript soll in /etc/init.d/umlRechnerName abgelegt werden
# Skript startet, stopt oder rebootet die UML-Instanz umlRechnerName
# Mit der Option backup kann ein onlinesnapshot der UML-Instanz
# erstellt werden.
# Skript erstellt 21.6.2004 M. Gonzenbach und S. Müller

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Konfigurationsdaten der UML-Instanz

NAME="umlRechnerName"
DESC="UML Instanz umlxx for Test"
KERNEL="/home/umladmin/kernel/deb-267uml-1"
MEMORY="mem=48M"
UMID="$NAME"
TAPDEV=tap_$UMID
TAPIP="127.0.1.1"
NETWORK="eth0=tuntap,$TAPDEV"
UMLIP="160.85.191.1"
DISKS="ubd0=/home/umladmin/disks/uml01bak
       ubd1=/home/umladmin/swapdisks/uml01"
CONSOLES="con0=fd:0,fd:1 con1=pts con=none"

#############################################################################

setup_tun() {     # Setzt das TAP-Device auf dem HOST auf und
                  # erstellt den Routingeintrag zur UML

      echo "/usr/bin/tunctl -d $TAPDEV"
      /usr/bin/tunctl -d $TAPDEV

      echo "/usr/bin/tunctl -t $TAPDEV -u umladmin -f /dev/net/tun"
      /usr/bin/tunctl -t $TAPDEV -u umladmin -f /dev/net/tun

      echo "/sbin/ifconfig $TAPDEV $TAPIP up"
      /sbin/ifconfig $TAPDEV $TAPIP up

      echo "/sbin/route del -net 127.0.0.0/8 dev $TAPDEV"
      /sbin/route del -net 127.0.0.0/8 dev $TAPDEV

      echo "/sbin/route add -host $UMLIP dev $TAPDEV"
      /sbin/route add -host $UMLIP dev $TAPDEV

      echo "bash -c 'echo 1 > /proc/sys/net/ipv4/conf/$TAPDEV/proxy_arp'"
      echo 1 > /proc/sys/net/ipv4/conf/$TAPDEV/proxy_arp

      echo "arp -Ds $UMLIP eth0 pub"
```

```
      arp -Ds $UMLIP eth0 pub

      echo "Das TAP-Device wurde erfolgreich installiert..."

}


set -e
case "$1" in

    start)
      echo -n "Starting: $NAME: $DESC"

      # Prüfe ob diese UML-Instanz bereits läuft
      if [ -f /home/umladmin/running_$UMID ]
      then
         echo "$UMID is already running. stop|restart|backup..."
         echo "Connect mit #screen -r <pid> zur Konsole von $UMID"
         exit 1
      fi

      setup_tun         #TUN/TAP für den User einrichten

      echo "Jetzt gehts los mit $UMID starten - connect mit #screen -r
            <pid> zum Loginpromt"
      #/usr/bin/screen -d -m -- su -c "$KERNEL $MEMORY $DISKS $NETWORK
                                      $CONSOLES $UMID" umladmin
      cd /home/umladmin/
      export TMP=/tmp/umlram
      screen -d -m -- su -c "$KERNEL $MEMORY $DISKS $NETWORK umid=$UMID
                             $CONSOLES" umladmin
      #echo "$KERNEL $MEMORY $DISKS $NETWORK umid=$UMID $CONSOLES"
      echo done
      echo "$UMID is running..." >running_$UMID

      echo "$UMID wurde erfolgreich gestartet..."
      echo "Connect mit #screen -r <pid> zum Loginpromt"
      echo "."
    ;;

    stop)
      echo -n "Stopping: $NAME: $DESC"
      sudo -H -u umladmin /usr/bin/uml_mconsole $UMID halt
      rm /home/umladmin/running_$UMID
      echo "$UMID wurde erfolgreich gestopt!"
      echo "."
    ;;

    restart)
      $0 stop
      sleep 5
      $0 start
    ;;

    backup)
      echo -n "Backup von: $NAME: $DESC"
```

```
      sudo -H -u umladmin /usr/bin/uml_mconsole $UMID stop
      sudo -H -u umladmin /usr/bin/uml_mconsole $UMID sysrq s
      su -c "cp -fra --sparse=always /home/umladmin/disks/$NAME
             /home/umladmin/backup/$NAME.bak" umladmin
      sudo -H -u umladmin /usr/bin/uml_mconsole $UMID go
      sudo -H -u umladmin /usr/bin/uml_mconsole $UMID quit
      sudo -H -u umladmin tar cfvj
/home/umladmin/backup/$NAME.bak.tar.bz2 /home/umladmin/backup/$NAME.bak
      #su -c "tar cfvj /home/umladmin/backup/$NAME.bak.tar.bz2
              /home/umladmin/backup/$NAME.bak"
      rm /home/umladmin/backup/$NAME.bak
      echo " Backup von $NAME wurde erfolgreich erstellt."
        ;;

    *)
      N=/etc/init.d/$NAME
      echo "Usage: $N {start|stop|restart|backup}" >&2
      exit 1
    ;;

esac
exit 0
```

9.2. Konfigurationsdateien

Host

/etc/apt/sources.list

/etc/network/interfaces

/etc/network/options

/etc/resolf.conf

/etc/fstab

/etc/hostname

/etc/hosts

/usr/src/linux/.config

/boot/grub/menu.lst

UML

/etc/apt/sources.list

/etc/network/interfaces

/etc/resolf.conf

/etc/fstab

/etc/inittab

/etc/securetty

/etc/hostname

/etc/hosts

/usr/src/linux-2.6.4/.config

9.3. Quellenverzeichnis

http://user-mode-linux.sourceforge.net

http://usermodelinux.org

http://www.debian.org

http://www.openoffice.de/linux/buch

http://www.user-mode-linux.org/~blaisorblade

http://uml.harlowhill.com

9.4. Abbildungsverzeichnis

9.5. Glossar

Backingfile	Urfilesystem (im Zusammenhang mit COW)
COW	Copy On Write
Debian	Linuxdistribution
DMZ	Demilitarisierte Zone
Ext2	Filesystem
Ext3	Journaling Filesystem (Erweiterung von Ext2)
Kernel	„Herzstück“ eines Betriebssystems
NFS	Protokoll für Datei- und Druckerservice
Reiser	Journaling Filesystem
SAMBA	Protokoll für Datei- und Druckerservice
SFTP	Secure File Transfer Protocoll (Erweiterung von FTP)
SKAS	Separate Kernel Address Space
Sparsefile	File mit fixer Grösse / effektive Grösse ist aber dynamisch
SSH	Secure Shell
TT	Tracing Thread
TUN/TAP	Treiber für die Realisierung eines virtuellen Netzwerkes
UML	User Mode Linux
Host	physischer Server auf welchem UML-Instanzen laufen
UML-Instanz	einzelner virtueller Server auf UML basierend
UML-Kernel	Kernel für eine UML-Instanz
UML-Provider	Webhostingprovider der UML-Instanzen anbietet