
# User-Mode Linux
## Seminar User-Mode Linux
## WS05/06

Tim Keupen
timbub@uni-koblenz.de

Universität Koblenz
25.11.2005

# Inhaltsverzeichnis

# 1 Einleitung

Diese Ausarbeitung entstand im Rahmen des Seminar Simulationen mit User-Mode Linux an der Universität Koblenz im Wintersemester 2005/06 unter der Leitung von Dipl.-Inform. Harald Dickel.

Im folgenden wird das Virtualisierungstool User-Mode Linux (UML) vorgestellt. Nach einer allgemeinen Einführung in die Funktionsweise werden die in UML vorhandenen Netzwerkmechanismen behandelt. Hierbei liegt das Augenmerk auf den von VNUML (Virtual-Network User-Mode Linux) tatsächlich verwendeten Transporttypen. VNUML ist eine Netzwerksimulation, die auf den Funktionalitäten von User-Mode Linux aufsetzt und wird in diesem Seminar verwendet um praxisbezogene Betrachtungen verschiedenster Themen aus dem Bereich Rechnernetze durchführen zu können. User-Mode Linux stellt hierbei die Grundfunktionalität, indem es virtuelle Maschinen erzeugt, die sich nach beliebiger Konfiguration, bis auf Performanz, wie beliebige Knoten in einer beliebigen Netztopologie verhalten.

## 1.1 Historie

Die Idee virtuelle Machinen zu verwenden ist nicht neu. Schon in den 70er Jahren konnten Mainframes von IBM sogenannte Gastbetriebssysteme starten. Ein klassisches Beispiel ist VM/370, eine virtuelle Maschine für IBM 370 Systeme:

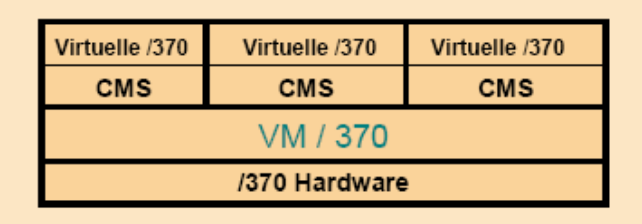


Abbildung 1: VM/370 von IBM

Dieses System gab jedem Benutzer den Eindruck er würde direkt und exklusiv die Hardware (Prozessor, Hauptspeicher, Sekundärspeicher etc) verwenden. Jeder Benutzer konnte dabei den vollständigen Befehlssatz ausführen. Als Betriebssystem lief für jeden Benutzer das Ein-Benutzer-System CMS. Somit konnte die IBM 370 simultan mit mehreren herkömmlichen Betriebssystemen betrieben werden, ohne dass ein komplexes Mehrbenutzerbetriebssystem dafür nötig war.[6]

Der Trend hin zu Desktop-PCs und der Folge daraus, dass jeder Benutzer einen eigenen PC auf seinen Schreibtisch bekommen sollte, ließ solche Konzepte einschlafen. In den letzten Jahren wurden die Ideen der 70er jedoch wieder neu belebt. Die anhaltend steigende Leistung der heute verfügbaren Hardware ermöglicht es sogar Multisysteme auf einem einzigen Gerät simultan zu betreiben. Spezielle Betriebsysteme wie z.B. Z/OS von IBM bilden eine Zwischenebe-

ne zur Hardware, auf der dann fast beliebige Betriebssysteme simultan laufen. Beim sogenannten Server-Hosting läuft auf einer physikalischen Maschine für jeden Kunden eine eigene virtuelle Maschine, die die gemieteten Dienste (z.B. Webserver oder Mailserver) anbietet. Untereinander dürfen sich diese virtuellen Maschinen nicht beeinflussen können. User-Mode Linux bietet hier auf Hardwareebene die Möglichkeit die Adressräume der Prozesse zu kapseln und ist somit für Systeme, die nicht im High-Perfomance Bereich angesiedelt sind, durchaus interessant.

## 1.2 Virtualisierungstechniken

Man unterscheidet drei grundlegend verschiedene Ansätze von Virtualisierungstechniken:[7]

**Hardware Emulators** Diese Programme emulieren Schritt für Schritte eine spezielle Hardware (Beispiele: Bochs, PearPC, coLinux und QEMU).

**Hardware Virtualization** Virtualisierung bezeichnet Methoden die es erlauben Ressourcen eines Computer aufzuteilen. Hierbei werden nur ausgewählte System Calls und Interrupts emuliert (Beispiele: VMWare, UML, plex86 und XEN).

**Limited Virtualization** vServer verstecken lediglich Teile des Betriebssystems. Es läuft nur ein Kernel, Systemaufrufe werden abgefangen und bezüglich der Zugriffsrechte etc. modifiziert.

Alle diese Techniken haben ihre Vor- und Nachteile. Systeme die eine komplette Hardware emulieren sind einerseits plattformunabhängig, andererseits auch sehr langsam, da jeder Systemaufruf eine zusätzliche Ebene, den Emulator, passieren muss. Als Beispiel sei an dieser Stelle BOCHS erwähnt, einer der ältesten verfügbaren Emulatoren. Es ist ein in C++ geschriebener x86 CPU-Emulator, die Performance ist vergleichsweise gering, es ist jedoch so portabel, dass es sogar möglich ist FreeDOS auf einem UltraSPARC zu booten.

Die beiden anderen Ansätze sind in der Regel betriebssystemunabhängig (Ausnahme: User-Mode Linux existiert momentan nur für Linux-Kernels), sind aber nur auf spezieller Hardware lauffähig, VMWare und UML laufen beispielsweise nur auf x86 Architekturen. Die Performanz ist nicht optimal, insbesondere wenn mehrere Instanzen virtueller Maschinen laufen und für jede eine entsprechende Hardware virtualisiert werden muss. Limited Virtualisation hingegen ist auf Geschwindigkeit und wenig Overhead optimiert. Durch die perfekte Einbindung in das Betriebssystem sind solche Systeme natürlich vollkommen abhängig von diesem. Hauptsächlich findet diese Technik Anwendung im Bereich der vServer, wenn also z.B. ein Mail- oder Webserver auf den stärksten verfügbaren PC im Rechnerpool, auf einem bereits laufenden Betriebssystem, aufgesetzt werden soll.

# 2 User-Mode Linux

„User-Mode Linux“ (UML) wird seit 1999 von Jeff Dike als Open-Source Porjekt entwickelt. Ursprünglich wurde das Projekt initiiert um das Debuggen des Linux-Kernels zu vereinfachen, indem ein in Entwicklung befindlicher Linux-Kernel im User-Space eines anderen Linux-Systems ausgeführt wird, sodass bspw. der GNU Debugger `gdb` und andere bei der Software-Entwicklung von regulären Linux-Programmen übliche Tools eingesetzt werden können.[5]

Verwendet wird hierzu ein Linux-Kernel, der nicht direkt auf die Hardware aufsetzt, sondern als Prozess in einem Hostsystem läuft. Wie jede Anwendung wird ein UML-Kernel gestartet, sobald man ihn braucht und kann ebenso leicht auch wieder beendet werden. Dabei läuft dieser Kernel im sogenannten „Userspace“, hat also nicht die volle Gewalt über das System. Dadurch hat der im Falle eines Fehlers oder Hacks entstehende Schaden so gut wie keine Auswirkungen auf das Hostsystem, also die GNU/Linuxumgebung in welcher der UML-Kernel gestartet wurde. Die folgende Abbildung verdeutlicht den Unterschied zum echten Linux-Kernel:

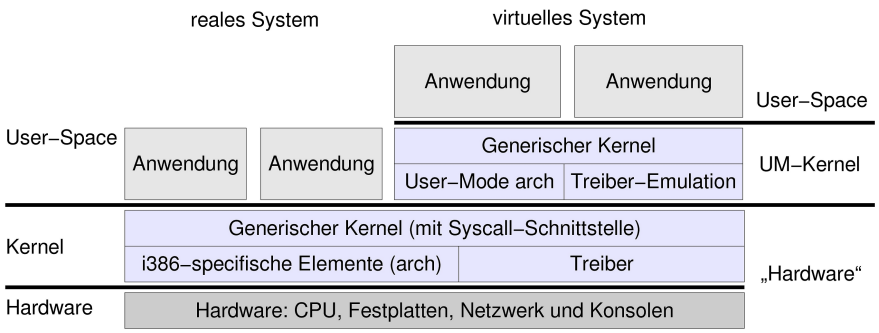


Abbildung 2: UML Kernel als Prozess

Während die linke Seite das bekannte Schichtenmodell eines Betriebssystems wiederspiegelt (Hardware, Kern, Prozesse), erweitert UML das Modell um eine weitere Schicht. Der UML-Kernel läuft auf Ebene der Prozesse über dem echten Kernel, erst darüber laufen die innerhalb der UML-Instanz nicht-privilegierten Anwendungen im Userspace. UML-Kernel und Anwendungen greifen also nicht direkt auf die Hardware zu, Systemaufrufe werden an den Hostkernel weitergereicht.

Vom Funktionsumfang lässt sich ein UML-Kernel vom echtem Betriebssystem-Kernel fast nicht unterscheiden. Einzig die Ausführung als neuer Prozess in einem bereits laufenden System bildet den Unterschied. Dabei arbeiten sowohl der Scheduler, als auch die Speicherverwaltung unabhängig vom Hostsystem, dieses wird lediglich für die Hardwareunterstützung genutzt. Insbesondere besitzt jeder gestartete UML-Kernel eine eigene Speicher-, Prozess- und Dateisystemverwaltung. Unterschiede im Kernelaufbau selbst sind seine soge-

nannte usermode-Architektur, die damit einhergehende Änderung der SysCall Schnittstelle und die Treibereinbindung (Gerätetreiber, z.B. Netzwerkdevices, sind nicht real verfügbar, sondern werden emuliert).[1]

Mit UML können mehrere virtuelle Linux-Systeme („Gäste“) gleichzeitig auf demselben physischen Linux-System („Host“) ausgeführt werden. Jeder Gast kann dabei unterschiedliche Versionen des Linux-Kernels (z.B. 2.4.28, 2.6.9-bb4) ausführen und auf verschiedenen Linux-Distributionen (z.B. SuSE Professional 9.2, Debian 3.0, Redhat,..) basieren.[5]

## 2.1 Virtuelle Maschinen

Eine UML-Instanz die folgende Eigenschaften besitzt, bezeichnet man als virtuelle Maschine (VM):

**Terminalfenster / Managementkonsole** Dient der Überwachung des Bootvorgangs und der späteren Steuerung und Konfiguration der VM. Selbst wenn die VM vollkommen autonom laufen soll (z.B. Selbstkonfiguration über Skripte und Netzwerkanbindungen), ist eine Managementkonsole nötig um Grundfunktionen wie `reboot` oder `halt` ausführen zu können. Neben einer Bootkonsole kann man in der „/etc/inittab“ noch weitere Konsolen definieren, die sich dann während des Bootvorgangs auf dem Hostsystem öffnen.

**Speicherbereich im RAM des Host** Der für die VM verfügbare Hauptspeicher wird im RAM des Host angelegt. Die Größe des virtuellen Hauptspeichers kann beim Aufruf über den Parameter “mem=“ angegeben werden. Der Wert sollte gut überlegt sein, da jede Instanz einer VM den entsprechenden Teil des Hauptspeichers im Hostrechner belegt. Das Memory-Management erfolgt hierbei in den Modi TT-Mode und SKAS-Mode (s.u.).

**Dateisystem** Jeder VM wird als Dateisystem eine Datei auf dem Host zugeordnet. Diese Datei ist eine Image-Datei, die eine komplette Verzeichnisstruktur (/bin, /etc/, /var usw.) einer regulären GNU/Linuxinstallation enthält. Aufgrund der fehlenden Partitionstabelle kann je nach UML-Kernel beim Bootvorgang der VM der Fehler `/dev/ubd/disc0: unknown partition table` ausgegeben werden. Da eine Image-Datei keine Partitionstabelle besitzt, kann dieser Fehler ohne weiteres ignoriert werden.

**Netzwerkschnittstellen** Diese Eigenschaft ist optional. Um eine Netzwerkverbindung zwischen der VM und dem Host, bzw. zwischen VM und VM zu ermöglichen ist die Konfiguration von Schnittstellen notwendig. User-Mode Linux bietet hierfür 7 Transporttypen (s.u.), je nach Typ sind hier die user-Rechte jedoch nicht mehr ausreichend, da Schnittstellen-Konfigurationen auf dem Host nur mit root-Rechten durchgeführt werden können.

Je nach Performanz des Host-Rechners können beliebig viele Instanzen virtueller Maschinen gestartet werden. Optimierungsmöglichkeiten liegen im Bereich der Speicherverwaltung des Host-Kernels (SKAS-Mode, s.u.) und in der Verarbeitung von Dateisystemänderungen (COW, s.u.). Sobald die Menge der Instanzen jedoch kleine Testszenarien übersteigt und bspw. größere Topologien simuliert werden sollen, ist ein großer Hauptspeicher im Hostsystem unerlässlich.

## 2.2 TT-Mode und SKAS-Mode

Standardmäßig startet jede VM im TT-Mode. Während des Startvorgangs wird das Vorhandensein des SKAS-Patches überprüft und bei Erfolg in diesem Modus gestartet (`Checking for the skas3 patch in the host...found / Checking for /proc/mm...found`).

### 2.2.1 TT-Mode

Der TT-Mode (Tracing-Thread-Mode) ist standard für nicht gepachte Hostkernel. Hierbei werden die normalen Prozess- und Speicherverwaltungstechniken verwendet, die in jedem normalen Linux Kernel enthalten sind. Teile des Hostkernel laufen jedoch standardmäßig im selben Speicherbereich wie der UML-Prozess und sind „writeable“. Dies hat ein Sicherheitsproblem zur Folge, da ein UML-Prozess Schreibzugriff auf Kernel-Daten hat und so ein Ausbruch aus dem user-mode in den Host möglich ist. Für Laborumgebungen zu Simulations- und Testzwecken ist dieser Nachteil jedoch unerheblich, solange keine sicherheitsrelevanten Dienste über das lokale Netz hinaus angeboten werden. Das Sicherheitsproblem lässt sich durch setzen der UML-Daten in den Jail-Mode (Daten sind nur noch „readable“) umgehen. Diese Lösung verlangsamt jedoch das UML-Konzept und ist insbesondere für die Anwendung im Bereich Honeypots (siehe Kap.4) ungeeignet, da die Kernel-Daten immernoch lesbar sind und somit ein Angreifer erkennen kann, dass er sich innerhalb einer VM befindet und nicht direkt auf dem Hostsystem.

Im Vergleich zum SKAS-Mode ist die Performanz des TT-Mode eher mäßig. Insgesamt sind für Systemaufrufe vier Kontextwechsel nötig (SysCall $\leftrightarrow$ TracingThread $\leftrightarrow$ Kernel).

### 2.2.2 SKAS-Mode

Der SKAS-Mode (Separate Kernel Address Space Mode) verwendet zusätzliche Memory-Management Features und erfordert somit einen Patch des Hostkernel. Der UML-Kernel läuft hierbei in einem komplett abgekapselten Prozess- und Adressraum, „bösartige“ Prozesse können somit nicht mehr auf den Kernelspeicher übergreifen. Durch die direkte Verwendung der modifizierten ptrace()-Funktion des Hostkernel ist dieser Modus schneller, da pro SysCall nur 2 Kontektwechsel nötig sind. Das SKAS-Patch ist momentan verfügbar für die Versionen 2.4.x und 2.6.x.[2]

## 2.3 Installation

Um User-Mode Linux installieren zu können, benötigt man einen Rechner mit x86 Architektur und einen Linux-Kernel ab Version 2.2.15.

Informationen, aktuelle Kernels und Dateisysteme sind auf der UML-Downloadseite zu finden: *http://user-mode-linux.sourceforge.net/dl-sf.html*. Dort ist es möglich fertige rpm- oder deb-Pakete zu laden, welche dann nur noch, entsprechend den distributionsabhängigen Schritten, installiert werden müssen.

### 2.3.1 Kernel

Alternativ zum fertigen Paket kann man auch einen standalone-Kernel mit einem passenden UML-Patch laden. Zusätzlich muss hier noch das Paket uml_utilities (uml_mconsole, uml_moo, uml_switch, uml_net, tunctl) installiert werden, um den Funktionsumfang von UML zu vervollständigen. Möchte man den Kernel selbst kompilieren, kann dies über die üblichen Schritte (Sourcen laden, kompilieren, konfigurieren) geschehen. Wichtig ist jedoch, kein normales i386-System für Intel/AMD Systeme, sondern, mit dem Argument `ARCH=um`, eine user-mode-Architektur zu erstellen.

### 2.3.2 Filesystem

Jede VM benötigt als Dateisystem eine Image-Datei auf dem Host, welche ein Rootfilesystem einer lauffähigen Linux-Distribution erhält. Fertige Rootfilesysteme für verschiedenste Distributionen finden sich auf der UML-Downloadseite (s.o.). Es ist jedoch auch problemlos möglich, z.B. aus der eigenen Verzeichnisstruktur ein Image zu erstellen und als Dateisystem zu verwenden. Wird dieses Dateisystem von mehreren UML-Instanzen gleichermaßen verwendet, ist es sinnvoll ein COW (Copy-On-Write) Dateisystem anzulegen, da ein einzelnes schnell mehrere Hundert Megabyte groß sein kann. Alle virtuellen Maschinen booten dann vom selben Filesystem, sämtliche Änderungen werden jedoch in eine VM-spezifische Datei (COW-File) geschrieben. Dies spart das mehrfache Anlegen des Dateisystems, hat aber den Nachteil, dass alle Systeme im Initialzustand identisch sind.

# 3 Anwendungsszenarien

User-Mode Linux wurde entwickelt um einzelne Kernel und Kerneländerungen in einfacher Weise testen zu können. Es wurde jedoch schnell deutlich, dass dieses Konzept durch geeignete Erweiterungen noch viel mehr bietet, heute kann User-Mode Linux mit zusätzlichen Funktionalitäten in den verschiedensten Umgebungen verwendet werden:[10]

**Testumgebung/Sandbox** Ohne viel Aufwand und Risiko ist es möglich einen neuen Kernel oder Treiber zu testen und zu debuggen, oder sich z.B. davon zu überzeugen, ob der Rechner mit dem neu gepatchten Kernel auch

booten würde. Administratoren bietet UML das sogenannte „disaster recovery practice“, bei dem versucht wird ein defektes System mittels der noch vorhandenen Funktionalität zu retten.
Die Bezeichnung Sandbox stammt hier tatsächlich von dem Wort „Sandkasten“, ein System welches isoliert vom Betriebssystem läuft und somit als Spielwiese genutzt werden kann um unterschiedlichste Dinge auszuprobieren.

**Virtuelles Hosting** Da es sich bei UML um einen vollwertigen Kernel handelt, ermöglicht es Hostingprovidern ein physisches System in mehrere, von einander unabhängige, virtuelle Maschinen aufzuteilen und so u.U. eine bessere Hardwareauslastung zu erzielen. Läuft der Host im SKAS-Mode, können auch sicherheitskritische Web-Services in UML Umgebungen eingebettet werden. Das bedeutet, dass z.B. Domain-, Mail- oder Webspaceserver als UML-Prozesse auf einem System laufen, durch die Kapselung der einzelnen Prozesse jedoch ein Hacker niemals das komplette Hostsystem blockieren kann, sondern allenfalls einen einzelnen Prozess darauf beeinflussen. Sollte der virtuelle Server tatsächlich gehackt werden, nimmt lediglich das entsprechende UML-System Schaden. Da dieser Prozess unabhängig neugestartet werden kann, spart man Wartungs- und Hardwareaufwand.

**Softwareentwicklung** Der Kernel innerhalb der virtuellen Maschine kann anders als das Hostsystem konfiguriert werden, bzw. es können Kernels beliebiger anderer Distributionen gestartet werden. Dies versetzt Softwareentwickler in die Lage, Software unter geänderten Bedingungen zu entwickeln und zu testen. Entwickler von netzwerkbasierten Programmen können ein virtuelles Netzwerk auf einem einzigen Rechner aufziehen, um ihre Applikation kostenschonend zu testen.

**Honeypots/Honeynets** Ein Honeypot ist ein System welches bewusst angreifbar gemacht wird, um somit Auskunft über die Strategien der Angreifer zu liefern. Durch die Besonderheiten von UML wird es möglich eine Vielzahl an virtuellen Honeypots auf einem physikalischen System laufen zu lassen und dabei sogar noch ein komplettes Netzwerk zu simulieren.

**Netzwerksimulation** Werden neben den virtuellen Maschinen auch Netze simuliert, können beliebige Topologien nachgebildet und untersucht werden (insbesondere Verhalten/Funktion von Protokollen). Da mit UML standardmäßig keine Netze emuliert werden können, eignet es sich nicht um Lastverteilungen zu untersuchen.

Viele weitere Szenarien sind denkbar. In welchen Umgebungen sich User-Mode Linux tatsächlich durchsetzen kann, wird sich in den nächsten Jahren zeigen. Als weitere Anregung sei angemerkt: man kann sogar innerhalb einer virtuellen Maschine einen weiteren UML-Kernel erstellen und starten.

# 4 Netzwerkkommunikation

Um die Funktionalität von User-Mode Linux zu erweitern sollen virtuelle Maschinen untereinander und mit dem Hostsystem kommunizieren können. Um dies zu ermöglichen sind in UML 7 verschiedene Transporttypen implementiert:

**ethertap** Erstellt Ethernet-Verbindungen, auch eine Anbindung zum Host ist möglich. Wird nur von Kernelversion 2.2 verwendet, da ab 2.4 obsolet.

**tun/tap** Standardtyp für Ethernet-Verbindungen, insbesondere wenn das Hostnetzwerk mit eingebunden wird. Bietet die beste Perfomance und ist sicherer als ethertap.

**multicast** Erstellt ein komplett virtuelles Netz. Eine physikalische Schnittstelle muss vorhanden sein, da die VMs über diese via multicast-Betrieb Pakete erhalten.

**switch daemon** Erstellt ein komplett virtuelles Netz, nur mit Hostanbindung wenn eine VM als Gateway konfiguriert ist. Anbindung an den Switch-Daemon wird über UNIX-Sockets erreicht.

**slip** Nur sinnvoll wenn explizit keine tun/tap-Verbindung gewünscht ist. Arbeitet auf Layer3 und kann nur das IP-Protokoll transportieren.

**slirp** Entspricht slip, läuft jedoch ohne root-Rechte. Die maximale Baudrate ist auf 115200 begrenzt.

**pcap** Erzeugt eine read-only Schnittstelle, geeignet für „monitoring“ und „sniffing“.

VNUML implementiert die virtuellen Netze mit Hilfe der Transporttypen *tun/tap* und *switch daemon*, wobei *tap* wiederrum verwendet wird um die UML-Instanzen an eine virtuelle Linux-Bridge (siehe Kap. 4.2) zu binden. Die *virtual_bridge* wird in Linux durch das bridge-Modul implementiert und läuft im Kernelspace (root-Rechte). Das Modul *uml_switch* läuft im Userspace. Da (standardmäßig) keine Schnittstellen auf dem Host angelegt werden müssen, sind für dieses Modul keine root-Rechte notwendig.

## 4.1 TUN/TAP

Das tun/tap-Modul muss vom Hostsystem bereitgestellt werden, ggf. ist der Kernel neu zu übersetzen (source: http://vtun.sourceforge.net). Die tun/tap-Devices rücken die Unix-Weltanschauung „Everything is a file“ wieder etwas gerade, denn bei Netzwerkschnittstellen trifft das normalerweise nicht zu, es gibt keine Gerätedatei /dev/eth0. Das Konzept von tun/tap ermöglicht einen Paketaustausch zwischen UML-Instanzen über eine Datei, das sog. devicefile. Pakete werden also nicht über ein physikalisches Medium übermittelt, sondern können zwischen einem Userspace-Programm und dem Kernel über das devicefile ausgetauscht werden. Dabei arbeitet tap mit Ethernet-Frames auf Layer 2 und tun mit IP-Paketen auf Layer 3.

Tun/Tap ist momentan für folgende Plattformen verfügbar: Linux Kernels 2.2.x, 2.4.x, 2.6.x; FreeBSD 3.x, 4.x, 5.x; Solaris 2.6, 7.0, 8.0.

### 4.1.1 TUN

Tun ist ein „virtual point-to-point network device“ und ist auf Schicht 3 des OSI-Modells implementiert. Der Tun-Treiber wurde als sogenannter „low-level-kernel-support“ für IP-Tunneling entwickelt. Er bietet für die Benutzeranwendung zwei Schnittstellen:

- **/dev/tunX** - devicefile
- **tunX** virtual - point-to-point interface

Eine Anwendung schreibt nun IP-Pakete nach /dev/tunX, der Kernel erhält dieses Paket von der tunX Schnittstelle. Umgekehrt kann der Kernel ein IP-Paket an die tunX Schnittstelle senden und die Anwendung kann dieses Paket von /dev/tunX lesen.[9]

### 4.1.2 TAP

Tap ist ein „virtual ethernet network device“ und ist auf Schicht 2 des OSI-Modells implementiert. Der Tap-Treiber wurde als sogenannter „low-level-kernel-support“ für Ethernet-Tunneling entwickelt. Er bietet für die Benutzeranwendung zwei Schnittstellen:

- **/dev/tapX** - devicefile
- **tapX** - virtual point-to-point interface

Eine Anwendung schreibt nun Ethernet-Frames nach /dev/tapX, der Kernel erhält diesen Frame von der tapX Schnittstelle. Umgekehrt kann der Kernel einen Ethernet-Frame an die tapX Schnittstelle senden und die Anwendung kann diesen Frame von /dev/tapX lesen.[9]
Diese Methode wird von User-Mode Linux verwendet. Über den Befehl `tunctl` aus dem uml_utilities-Modul lassen sich diese Schnittstellen für jeden Benutzer anlegen: `tunctl -u uid device`.

## 4.2 Linux Bridging

Eine Bridge leitet Verkehr auf Schicht 2 des OSI-Modells zwischen Netzsegmenten weiter. Sie wird hauptsächlich eingesetzt, um ein Netz in verschiedene Kollisionsdomänen aufzuteilen. Somit kann die Last in großen Netzen vermindert werden, da jeder Netzstrang nur die Pakete empfängt, deren Empfänger sich auch in diesem Netz befindet.[4]

Die Netze die durch eine Bridge verbunden sind, bilden ein größeres logisches Netz. Die Arbeit die durch die Bridge verrichtet wird, soll auf IP-Ebene völlig transparent bleiben, in einem `traceroute` kommt somit keine Bridge vor.

Im Linux-Kernel ist **bridging** seit Version 2.4 vollständig implementiert. Dies ermöglicht das Starten einer virtuellen Bridge innerhalb eines Linux-Hostsystems. Da die „virtual_bridge“ innerhalb des Host als Schnittstelle repräsentiert wird, sind für die Ausführung der Kommandos root-Rechte nötig. Um die Algorithmen administrieren zu können, wird das Paket „bridge-utils“ benötigt. Darin enthalten ist das Kommando `brctl`:

**brctl addbr *bridgename*** Erzeugt eine logische Instanz einer Bridge mit dem Namen *bridgename*. Die Bridge kann als Container für die ihr hinzugefügten Schnittstellen interpretiert werden, die dann alle in einem größeren logischen Netz liegen. Jede Instanz wird durch ein neues Netzwerk-Interface im Host repräsentiert.

**brctl delbr *bridgename*** Beenden und Löschen einer Instanz.

**brctl addif *bridgename device*** Fügt die Schnittstelle mit dem Namen *device* zur Bridge *bridgename* hinzu. Nach kurzer Zeit lernt die Bridge die MAC-Adresse und kann Pakete gezielt weiterleiten.

**brctl delif *bridgename device*** Entfernt die Schnittstelle mit dem Namen *device*.

**brctl show** Zeigt eine Zusammenfassung aller aktiven Instanzen und der aktuell eingebundenen Schnittstellen.

**brctl showmacs *bridgename*** Zeigt alle der Bridge bekannten MAC-Adressen und deren Timer.

**brctl stp *bridgename* on/off** Aktiviert/Deaktiviert das Spanning Tree Protocol (STP), welches benötigt wird um bei redundant vernetzten Bridges Endlosschleifen zu vermeiden.

Das folgende kurze Beispiel macht aus einem Linux-Rechner mit 2 Netzwerkkarten schon eine voll funktionsfähige Bridge:

```
# ifconfig eth0 0.0.0.0 promisc up
# ifconfig eth1 0.0.0.0 promisc up
# brctl addbr mybridge
# brctl addif mybridge eth0
# brctl addif mybridge eth1
# ifconfig mybridge up
```

Die Schnittstellen eth0 und eth1 lauschen im „promiscuous-Mode“ auf allen im Netz befindlichen Verkehr, auch auf Verkehr der nicht expilizit für diese Schnittstelle bestimmt ist. Der letzte Befehl startet die *virtual_bridge* und Verkehr wird auf Schicht 2 zwischen den Schnittstellen weitergeleitet. Die Bridge hat in diesem Fall keine eigene IP-Adresse und ist somit gegen Hacker-Angriffe via TCP/IP geschützt.[8] Da sich die Bridge wie eine normale Schnittstelle verhält, lässt

sie sich mit dem Kommando "`# ifconfig mybridge 192.168.1.1 netmask 255.255.255.0 up`" als Teil des Netzwerks definieren und ist für den Benutzer zugänglich. Zum Entfernen der Bridge führt man die entsprechenden Lösch-Kommandos in umgekehrter Reihenfolge aus.

### 4.2.1 Linux Bridging in VNUML

In VNUML sind in der Version 1.5 standardmäßig alle in der Simulation erzeugten Schnittstellen über solche *virtual_bridges* verbunden. Dazu wird für jede Schnittstelle einer VM ein tap-device (ohne IP-Adresse) auf dem Host angelegt, welcher dann an die für das jeweilige Netz erzeugte *virtual_bridge* gebunden wird. Durch diesen „Trick" muss der Verkehr zwischen den virtuellen Maschinen nicht auf IP-Schicht durch den Host geroutet werden: Ist auf dem Host IP-Forwarding aktiv, würde die reine Anbindung der VM via tap-devices genügen um eine Kommunikation zu anderen VMs herzustellen, da Vehrker durch den Host an die jeweilige Schnittstelle weitergeitet würde, sofern die erzeugten tapX Schnittstellen auf dem Host eine IP-Adresse besitzen. Dies würde jedoch auf Schicht 3 einen zusätzlichen Hop bedeuten (VM1 - Tap1 - VM2). Aus diesem Grund verlagert VNUML die Vernetzung auf Schicht 2 und verwendet dafür eine direkte Anbindung der tap-devices an die oben vorgestellte „virtual_bridge" (diese Anbindung kann als ein „einstecken" eines Ethernet-Kabels an ein Layer2-Gerät interpretiert werden). Da tap-devices Ethernet-Frames verwenden, ist über die Bridge nun eine vollständige Verbindung auf Schicht 2 hergestellt.

Ein `traceroute` zeigt, dass zwei virtuelle Maschinen die via tap-devices an eine „virtual_bridge" gebunden sind, nun auf Schicht 3 direkt (ohne zusätzlichen Hop) verbunden sind. Da beliebig viele Instanzen virtueller Bridges erzeugt und bis zu 255 (Kernel 2.4) bzw. 1023 (Kernel 2.6) Schnittstellen an jede Bridge gebunden werden können, ist die Simulation (fast) beliebiger Ethernet-Topologien möglich.

Der Aufbau der virtuellen Netze ist für den Benutzer transparent und wird durch den VNUML-Parser anhand der XML-Spezifikation übernommen. Externe Verbindungen sind mit VNUML nur über diesen Typ möglich (`<net name='Net0' mode='virtual_bridge' external='eth1'/>`). Als zweite Variante bietet VNUML den Transporttyp switch daemon, mit dem ein komplett virtuelles Netz mit Hilfe des Moduls *uml_switch* erstellt wird (`<net name='Net1' mode='uml_switch'/>`). Eine Anbindung an das Host-Netzwerk ist zwar via tap-devices auch beim *uml_switch* möglich, wird von VNUML aber nicht unterstützt um den Vorteil der user-Rechte nicht zu verlieren.

## 4.3 Switch Daemon

Das Modul *uml_switch* ist ein in Software implementierter Ethernet-Switch. Es läuft im Userspace, erreicht durch die nötigen Kontextwechel in den Kernelspace jedoch nicht die Performanz der *virtual_bridge*. Der Daemon wird als Benutzer mit user-Rechten über die Kommandozeile gestartet (z.B. `$ uml_switch -unix /tmp/switch1.sock`). Dabei muss ein UNIX-Socket spezifiziert werden. Über den Paramter `-hub` ist eine Konfiguration als Hub möglich und über `-tap tap0` eine Anbindung an den Host, wobei „tap0“ hier ein vorkonfiguriertes tap-device sein muss. Der Aufbau eines virtuellen Switch-Netzwerks erfolgt in 2 Schritten:

- switch daemon (mit Parametern) starten und einen UNIX-Socket spezifizieren
- UML-Kernels booten unter Angabe des Transporttyps `daemon` (das Argument „socket“ ist dabei der Dateiname des spezifizierten UNIX-Sockets)

Soll eine VM an einen *uml_switch* gebunden werden, muss beim Start folgender Paramter angegeben und spezifiziert werden: `ethn=daemon,ethernetaddress, socket type,socket`. Die Kurzform `eth0=daemon` genügt hier auch, in diesem Fall wird die virtuelle Schnittstelle „eth0“ an den Standardsocket gebunden. Läuft mehr als ein *uml_switch* muss der entsprechende Socket angegeben werden.

Stürzt ein switch daemon ab, müssen alle angebundenen VMs neugestartet werden. Da der Daemon im Userspace läuft, ist er kritischer als die *virtual_bridge* und somit weniger geeignet für Hochverfügbarkeitssysteme.[3]

## 4.4 Virtuelle Schnittstellen erzeugen

Um innerhalb einer virtuellen Maschine eine Netzwerkschnittstelle zu simulieren, muss diese beim Start des User-Mode Kernels über Boot-Parameter spezifiziert werden. Um eine Schnittstelle ethN zu erzeugen, müssen der Transporttyp und die nötigen Argumente angegeben werden:

**multicast:** `eth<n>=mcast`
**tun/tap:** `eth<n>=tuntap,<device>,<ethernet address>,<IP address>`
**ethertap:** `eth<n>=ethertap,<device>,<ethernet address>,<tapIP address>`
**switch daemon:** `eth<n>=daemon,<ethernet address>,<sockettype>,<control_socket>, <data_socket>`
**slip:** `eth<n>=slip,<slip IP>`
**slirp:** `eth<n>=slirp,<ethernet address>,<slirp path¿`
**pcap:** `eth<n>=pcap,<host interface>,<filterexpression>,<option1>,<option2>`

Die Angabe des Transporttyps ist zwingend. Die restlichen Argumente können weggelassen werden und werden vom jeweiligen Modul mit Standardwerten (sockettype) bzw. generierten Werten (ethernet address) belegt.

## 4.5 Beispiele

Folgender Aufruf

```
host> linux  udb0=root_fs  eth0=tuntap,,,192.168.2.200  umid=uml1
```

startet einen UML-Kernel namens „linux“ und bootet von einem Dateisystem names „root_fs“, welches sich im selben Verzeichnis befindet. Weiterhin wird eine virtuelle Schnittstelle „eth0“ erzeugt, die über den Transporttyp tun/tap mit dem Host verbunden ist. Da kein tap-device angegeben wird, wird auf dem Host das nächste freie tapX erzeugt. Die Ethernet-Adresse wird automatisch generiert und die IP-Adresse des tap-devices im Host soll „192.168.2.200“ lauten. Die IP-Adresse der Schnittstelle eth0 kann hier nicht definiert werden, sie muss z.B. durch ein `ifconfig` innerhalb der VM durch den Benutzer angegeben werden. Durch den letzten Parameter bekommt die VM den Namen „uml1“.

Folgender Aufruf startet zwei virtuelle Maschinen, die via tap-devices mit dem Host verbunden werden:

```
host> linux eth0=tuntap,,,192.168.2.200 umid=uml1
uml1> ifconfig eth0 192.168.2.201 up

host> linux eth0=tuntap,,,192.168.2.210 umid=uml2
uml2> ifconfig eth0 192.168.2.211 up
```

Innerhalb der VM muss mit dem Kommando `ifconfig` die entsprechende Schnittstelle gestartet werden. Ist innerhalb des Hostsystems *ip-forwarding* aktiv, können uml1 und uml2 miteinander kommunizieren. Wird nun auf dem Host eine „virtual_bridge“ angelegt (`brctl addbr beispiel`) und die erzeugten tapdevices daran gebunden (`brctl addif beispiel tap0`, `brctl addif beispiel tap1`), können uml1 und uml2 auf Ethernet-Ebene kommunizieren. Sie sind somit auf IP-Ebene direkt verbunden und Verkehr muss nicht mehr durch den Host geroutet werden.

Folgender Aufruf startet einen Switch1, einen Switch2 und drei daran gebundene virtuelle Maschinen:

```
$ uml_switch -unix /tmp/switch1.sock
$ uml_switch -unix /tmp/switch2.sock

$ linux umid=uml1 eth0=daemon,,unix,/tmp/switch1.sock
$ linux umid=uml2 eth0=daemon,,unix,/tmp/switch1.sock eth1=daemon,,unix,/tmp/switch2.sock
$ linux umid=uml3 eth0=daemon,,unix,/tmp/switch2.sock
```

Uml1 ist dabei mit Switch1 verbunden, uml2 ist mit Switch1 und Switch2 verbunden und uml3 ist mit Switch2 verbunden. Die Kommunikation zwischen den VMs erfolgt nun direkt über den switch daemon.

# 5 Fazit

User-Mode Linux ist ein Virtualisierungskonzept, dass es ermöglicht speziell konfigurierte Linux-Kernel in einem laufenden Linux-System als Userprozess zu starten. Dabei wird die Hardware für jede virtuelle Maschine „virtualisiert“ (Netzwerkschnittstellen, Dateisystem als Image-Datei), so dass ein Benutzer innerhalb der VM den Eindruck hat, das System sei physikalisch tatsächlich vorhanden. Die verfügbaren Ressourcen sind jedoch durch den Host bestimmt, da die Hosthardware auf die Maschinen aufgeteilt wird (RAM, CPU-Last, Plattenplatz etc).

Im Bereich der Netzwerksimulation wird UML in mehreren Konzepten verwendet (z.B. VNUML,NetKit). Diese Konzepte erlauben es jedoch nicht Last- bzw. Performance-Messungen durchzuführen. Dies liegt zum einen daran, dass die verwendeten Transporttypen garkeine Bandbreiten simulieren und zum anderen ist es nicht möglich den virtuellen Maschinen ein (zeitlich korrektes) Verhalten, dass dem eines realen physikalischen Rechners entspricht, zuzusprechen. Insbesondere wenn mehrere VMs simultan betrieben werden, kann der Standard-Linux-Scheduler nicht sicherstellen, dass für alle dauerhaft die gleichen CPU-Ressourcen zu Verfügung stehen.

User-Mode Linux eignet sich jedoch sehr gut, um Labor- bzw Testumgebungen zu erstellen und anhand dieser die Funktion bzw. das Verhalten von Software oder Netzwerkprotokollen zu untersuchen.

# Literatur

[1] „Anleitung zur Installation des Netzwerksimulators VNUML“. Andre Volk, Tim Keupen. Uni Koblenz, Projektpraktikum Routingssimulation, SS2005.

[2] User-Mode-Linux HOWTO. „http://user-mode-linux.sourceforge.net/UserModeLinux-HOWTO.html“.

[3] Virtual Networking with User-Mode Linux. Kuthonuzo Luruo. März 2005.

[4] Bridge (Netzwerk) „http://de.wikipedia.org/wiki/Bridge_(Netzwerk)“. Wikipedia.de - Die freie Enzyklopädie.

[5] Virtuelle Testumgebungen mit User Mode Linux. Armin M. Warda, Postbank Systems AG. IBM pSeries, AIX Linux Technical University, München, 26.-29. Oktober 2004.

[6] UMLinux als Sandbox. Kerstin Buchacker, Hans-Jörg Höxer und Volkmar Sieh. http://www3.informatik.uni-erlangen.de/ Persons/hshoexer/publications/bsi2003.pdf Deutscher IT-Sicherheitskongreß des BSI, 2003.

[7] Gentoo News. Linux Virtualisierungstechniken, 13.12.2004 http://www.gentoo.org/news/de/gwn/20041213-newsletter.xml.

[8] Linux Ethernet Bridging. „http://bridge.sourceforge.net“.

[9] Universal TUN/TAP Driver Page. „http://vtun.sourceforge.net/tun“.

[10] Einführung in User Mode Linux. Cargal 2004.