Eigener Kernel und eigenes Dateisystem für User-Mode-Linux

Matthias Korn

2. Dezember 2005

Inhaltsverzeichnis

1 Motivation

Im Rahmen des Seminars Netzwerksimulation mit VNUML (Virtual Network with User Mode Linux) sollte die Frage geklärt werden, wie man die einzelnen Instanzen der virtuellen Systeme sich den Bedürfnissen anpassen kann, die man zur jeweiligen Simulation benötigt. Dabei reichen die Vorgegebenen Dateisysteme und Kernel des Projekts leider nicht immer aus, so dass es eventuell notwendig wird sich eigene Zusammenstellungen zusammenzubauen. Diese Arbeit soll nun die Grundlegenden Informationen liefern, wie man dabei vorzugehen hat.

2 Einführung

Der folgende Abschnitt soll einen kleinen Einblick in die Welt des Linux-Kernels und seine Verwaltung verschiedener Dateisysteme geben.

2.1 Der Kernel

Die Aufgabe des Linux-Kernels besteht allein darin die Prozesse des Systems zu verwalten und zwar Ausführungszeit und Speicherzugriff und ihnen eine einheitliche Schnittstelle zum Zugriff auf das System, Geräte und Dateien zur Verfügung zu stellen. Daraus ergibt sich die etwas eigenartige Situation, dass ein Linux zwar ohne den Kernel weder Lauffähig noch ein Linux ist, aber nur der Kernel zwar ein Linux ausmacht aber trotzdem auch weder ein Linux noch lauffähig sein kann.

2.1.1 Bootvorgang

Nachdem der Kernel in den Arbeitsspeicher des Systems zum Beispiel durch einen Bootmanager geladen wurde initialisiert er alle Speicherbereiche, die er zur Erledigung seiner Arbeit benötigt und versucht den ersten Prozess zu starten. Dieser Prozess hat den Namen *init* und wird in den Ordnern */sbin*, */etc* und */bin* gesucht, kann ihm aber auch durch die Option *init=<init-Programm>* vor dem Start übergeben werden. Von nun an tut der Kernel nur noch das, was er am besten kann und überlässt diesem Prozess dem er normalerweise die Prozessnummer 1 zuordnet alles weitere um aus dem Datengewirr ein lauffähiges Linuxsystem zu machen.

2.2 Dateisysteme

Der Linux-Kernel ist heutzutage in der Lage so ziemlich alle Arten von Dateisystemen, die die Computerwelt so zu Tage gebracht hat zu lesen und

viele davon auch zu beschreiben. Allerdings ist diese Aussage nicht ganz korrekt. Denn eigentlich müsste man sagen, dass es für die Dateisysteme Kernel-Treiber gibt, die in der Lage sind es dem Kernel so aufzubereiten, dass er sie in sein virtuelles Dateisystem einhängen und so den unter ihm verwalteten Prozessen zur Verfügung stellen kann.

2.2.1 Das virtuelle Dateisystem

Unter Unix und damit auch unter Linux gibt es nicht wie unter Windows das Konzept der Laufwerke. An ihrer Stelle gibt es einen so genannten Dateibaum mit einer Wurzel „/" und den hierarchisch darunter liegenden Ordnern und Dateien, wobei jeder Ordner, wie bei Windows, dazu dient die Wurzel eines ihm unterliegenden Teilbaums zu sein. Des weiteren wird versucht alle Funktionalitäten, die eine Lese- und / oder Schreiboperation benötigen, durch die Abstraktion einer Datei zu realisieren. So kommt es, dass man das Abspielen von Musik dadurch erreichen kann, in dem man den Inhalt einer Wave-Datei in die Datei */dev/dsp* kopiert. Oder hat man eine Webcam in sein System erfolgreich integriert kann ein Video-Chat-Programm die Datei */dev/v4l/video0* öffnen um den Aufgenommen Videostrom über das Netzwerk zu schicken.

Intern arbeitet der Linux-Kernel natürlich nicht über eine Verwaltete Liste von Namen, der ihm sagt welcher Treiber nun für die Aktuelle „Datei" zuständig ist sondern Speichert sein den Dateibaum als Baum von I-Nodes (Index-Knoten). Innerhalb eines I-Node wird der Typ der Datei (reguläre Datei, Ordner, Gerät oder symbolische Verknüpfung), die Zugriffsrechte, eine eindeutig identifizierende Nummer gespeichert und ein Verweis auf die Daten gespeichert. Will man nun auf ein reales Dateisystem zugreifen, so muss dieses in das virtuelle Dateisystem „eingehängt" (*mount*) werden. Dies funktioniert nur unterhalb eines leeren Ordners, da ein Ordner, ebenfalls eine Datei, die Beziehung zwischen Namen und I-Node enthält. Während des einhängens wird der I-Node des Ordners, in dem das Dateisystem dann zu finden ist, mit den dateisystemspezifischen Funktionen des Dateisystemtreibers Verbunden. Damit ein späterer Zugriff auf die Daten möglich wird muss der Treiber also die Funktionalität des virtuellen Dateisystems auf die das reale übertragen.

2.2.2 Was ist das root-Filesystem?

Beim Start des Systems ist dieses Virtuelle Dateisystem in der Regel leer, d.h. das Wurzelverzeichnis enthält keine Einträge. Es gibt also keine Möglichkeit für den Linux-Kernel irgendein Initialisierungsprozess zu starten.

Daher wird ihm über die Option *root=<root-fs>* die Position eines Dateisystems übergeben das er an der Wurzel einhängen kann und welches dann Sinnvollerweise alle nötigen Dateien enthält um den Bootvorgang zu beginnen. Da der Kernel erst danach in der Lage ist Zusatzmodule zu laden, sofern sie in diesem Dateisystem vorhanden sind und der initialisierende Prozess den entsprechenden Systemaufruf zum laden durchführt, muss die nötige Funktionalität zum lesen des übergebenen Dateisystems fest eingebaut haben.

Das root-Filesystem muss also prinzipiell nur den Initialisierungsprozess enthalten und alles was dieser benötigt um weitere Dateisysteme in das nun initialisierte Virtuelle Dateisystem einzuhängen um so den Systemstart zu vollenden. In der Regel enthält es allerdings alles, was zu einem lauffähigen System der gewünschten Aufgabe notwendig ist. Das Konzept eines minimalen root-Filesystems findet man jedoch auch in der Realität, zum Beispiel bei Slimclients oder bei jeder Linux-Distribution in der so genannten *initrd* (initialen Ramdisk).

Bei initialen Ramdisk wird dem Kernel über die Option *initrd=<initrd image>* ein Image (Speicherabbild) eines minimalen root-Filesystems übergeben, welches er in den Arbeitsspeicher kopiert und dann als root-Filesystem einhängt. Der Initialisierungsprozess wird dann in diesem gesucht und muss nun dafür sorgen, dass alle Vorbereitungen getroffen werden um das richtige root-Filesystem zu laden, dazu wird die Ramdisk wieder ausgehängt und das eigentliche root-Filesystem an ihrer Stelle an der Wurzel des virtuellen Dateisystems eingehängt, und um den Speicher wieder freigeben zu können. Nur der Vollständigkeit halber sei erwähnt, das ein Linux-System auch vollständig innerhalb einer solchen initialen Ramdisk lauffähig sein kann da sie ja für den Linux-Kernel die gleiche Funktionalität übernimmt wie das root-Filesystem. Als Beispiel sei die SuSE-Systeminstallationsumgebung genannt.

2.2.3 Sparse Files

Bei einer Festplatte werden die Daten in Blöcken einer festen Größe eingeteilt auf die dann mittels einer eindeutigen Blocknummer zugegriffen werden kann. Auf Grund dieses Umstandes gehören Festplatten unter Linux auch in die Kategorie der blockorientierten Geräte. Jeder Block ist historisch bedingt 512 Bytes groß. Innerhalb der meisten Dateisysteme werden mehrere dieser Blöcke zu einem logischen Block zusammengefasst. Dieser Logische Block (1kB-16kB) ist die kleinste Speichereinheit, die über das Dateisystem adressiert werden kann. Innerhalb solch eines Blockes befinden sich dann entweder Verwaltungsdaten oder Daten einer Datei. Dabei sieht man, dass Dateien die kleiner als ein logischer Block sind Overhead

erzeugen und Dateien die größer sind mehrere Blöcke einnehmen.

Einige Dateisysteme (z.B. ext2/3, reiserFS, XFS, aber auch NTFS) nutzen die Eigenschaft von in Blöcken zerstückelten Dateien in soweit aus, dass sie nicht nur die Datei über die Festplatte hinweg verteilen (fragmentierte Dateien) können, sondern, sollte es vorkommen, dass Block nur mit Nullen zu füllen ist, dieser gar nicht auf der Festplatte reserviert wird und damit anderen Dateien zur Verfügung steht. Erkennt der Treiber des Dateisystems den letzten Vorfall und nutzt entsprechend die Fähigkeiten des Dateisystems, so spricht man von einer spärlichen (sparse) Datei. Wenn man diese Dateien liest und kommt an eine Stelle, zu der es keinen logischen Block gibt in dem die Daten gespeichert sind muss der Dateisystemtreiber Nullen zurückliefern. Will man jedoch in solch einem Bereich schreiben muss er einen freien Block auf der Festplatte finden und der Datei zuordnen bevor der Schreibvorgang beginnen kann. Auf diese Weise ist es möglich mehrere GB große Dateien auf der Festplatte zu haben, die aber reell nur wenige kB an Platz verbrauchen. Unter Unix-Systemen kann man übrigens mit dem Kommando *ls -s* die Dateinamen und ihren wahren Blockverbrauch ansehen.

2.2.4 Zugriff auf Dateisysteme

Wie bereits erwähnt müssen zusätzliche Dateisysteme in den Dateibaum eingehängt werden. Das dafür notwendige Kommando ist der *mount* Befehl. Diesem übergibt man als ersten Parameter den Pfad zur Gerätedatei, die das einzuhängende Dateisystem enthält und als zweiten den Pfad zu dem Ordner unter dem die Dateien dann zu finden seien sollen. Nun wird versucht das Dateisystem zu identifizieren und dann einzuhängen. Sollte die automatische Erkennung fehlschlagen kann über die Option *-t <type>* dem *mount* Befehl der Typ mitgeteilt werden. Eine Liste aller aktuell vom Kernel unterstützen Dateisysteme ist in der Datei */proc/filesystems* zu finden.

2.2.5 Dateien als Festplatten

Wie schon erwähnt entspricht der Zugriff auf Geräte und damit auch auf Festplatten dem Zugriff auf Dateien. Da liegt es nahe Dateien wie Festplatten zu behandeln. Um nun auch dem Dateisystemtreiber davon zu überzeugen, dass er es mit einer Festplatte zu tun hat gibt es das loop-Device. Damit man nun also den Inhalt einer Datei als Dateisystem einhängen kann aktiviert man diese Gerät über die Option *-o loop* des *mount* Befehls. Natürlich müssen die Daten in der Datei dem Dateisystem entsprechend organisiert sein. Dazu muss es sich bei der Datei um ein Speicherabbild

einer echten Festplatte (oder CD, oder ...) handeln oder man muss in einer leeren Datei erst eine solche Struktur erzeugen. Dazu kann man die gleichen Programme wie zum formatieren richtiger Festplatten benutzen.

3 Der User-Mode-Kernel

An sich verhält sich der User-Mode-Kernel für die unter ihm laufenden Prozesse wie ein normaler Linux-Kernel. Die Bootkonsole findet man als Standardausgabe der Konsole in der der User-Mode-Kernel als normales ausführbares Programm gestartet wurde. Für alle weiteren virtuellen Konsolen startet der User-Mode-Kernel ein xterm (X-Window-Terminal), steht ihm kein X-Window zur Verfügung muss man mit Hilfe eines pseudo Terminals auf sie zugreifen. Zur Emulation von blockorientierten Geräten kann man den User-Mode-Kernel durch die Option *ubd*n=<*imagefile*> dazu veranlassen ein blockorientiertes Gerät zu erzeugen auf das dann innerhalb des virtuellen Systems über */dev/ubd/*n zugegriffen werden kann. Dabei verwendet man üblicherweise Speicherabbilder realer Partitionen oder Festplatten kann aber natürlich auch echte blockorientierte Geräte an das virtuelle System übergeben, unter der Voraussetzung der ausführende Benutzer hat auch Zugriffsrechte auf diese Geräte. Auch Netzwerkgeräte können in die virtuellen Systeme integriert werden. Da die Konfiguration eines Netzwerkes jedoch die Stärke von VNUML ist, muss hier nicht weiter darauf eingegangen werden.

Auf der Seite des Hosts gibt sich der User-Mode-Kernel als ausführbares Programm. Da er aber trotzdem noch ein Kernel ist finden ständig Kontextwechsel statt, die durch ihr immer wieder auftretendes Überprüfen der Zugriffsrechte die Ausführung innerhalb der User-Mode-Umgebung stark verzögern. Des weiteren können Abstürze von Prozessen innerhalb der User-Mode-Umgebung zu einem Absturz des User-Mode-Kernels führen, da nicht wie beim richtigen Kernel die Prozesse im selben Adressraum wie der User-Mode-Kernel laufen. Als Lösung dieser Probleme bieten die User-Mode-Entwickler eine Anpassung an den normalen Kernel an, die es ermöglicht dem User-Mode-Kernel einen eigenen gesonderten Adressbereich zuzuordnen, weshalb sie sie auch als SKAS-Mode (Separate Kernel Address Space) bezeichnen. Durch diese Maßnahme wird die Ausführung von Prozessen nicht nur deutlich schneller, sondern auch wesentlich sicherer. Ein weiterer Vorteil der sich aus dieser Betriebsart ergibt, ist, dass es nun auch für böswillige Prozesse nicht mehr erkennbar ist, dass sie nur in einer virtuellen Linux-Umgebung befinden.

3.1 Der eigene Kernel

Die Zeiten, dass man nach der Installation von Linux erstmal seinen eigenen Kernel bauen muss sind spätestens seit der Einführung des Linux-Kernels in der 2.6er Version in die großen Distributionen vorbei. Auch das Verfahren bei dem Bedarf eines Neubaus haben sich inzwischen zumindest für den englischkundigen Anwender deutlich verbessert.

3.1.1 Woher bekomme ich die Quellen?

Natürlich ist es dem erfahrenen Anwender jederzeit möglich sich die Aktuellen Kernel-Quellen direkt bei den Entwicklern `http://www.kernel.org/` herunterzuladen. Jedoch bieten die großen Distributionen innerhalb ihrer Paketverwaltung ein Paket mit dem Namen *Kernel-Source* an. Dadurch wird nicht nur die Installation wesentlich angenehmer, sondern man erhält auch gleich alle Distributionsspeziefischen Anpassungen. Nach dessen Installation findet man normalerweise im Ordner */usr/src/linux-<version>* entweder ein gepacktes TAR-Archiv der Quellen oder sogar schon alle nötigen Dateien in entpackter Form.

3.1.2 Konfiguration

Nachdem man nun die Quellen in einen Ordner Entpackt hat bleibt nun auch dem Grafik orientierten Anwender der Weg zu einer Kommandokonsole nicht erspart. Nachdem er mittels des Kommandos *chdir* in den entsprechenden Ordner gewechselt ist führt ihn ein *make oldconfig* und danach ein *make xconfig* wieder zu seiner geliebten grafischen Benutzeroberfläche zurück. Der erste Befehl diente dem laden der alten Konfiguration was normalerweise zu den Einstellungen führt, die die Distributoren für den Bau ihres Kernels verwendet haben.

Will man statt einen Systemkernel einen User-Mode-Kernel erstellen, ist darauf zu achten, das man den beiden *make*-Anweisungen ein *ARCH=um* hinzufügt. Man wechselt also wieder mittels *chdir* in den Ordner in dem man die Quellen entpackt hat und landet nach *make oldconfig ARCH=um* und *make xconfig ARCH=um* wieder in der grafischen Konfigurationsoberfläche.

Nun kann man sich den Hilfetexten entsprechend bis ins kleinste seinen Kernel anpassen, oder nur darauf achten, dass alle Optionen für den User-Mode-Betrieb entsprechend aktiviert sind. Dazu zählt beim System-Kernel die Aktivierung der SKAS Funktionalität und die TUN/TAP Netzwerkunterstützung. Beim User-Mode-Kernel sollte man natürlich alles aktivieren

was man für seine Simulation verwenden möchte und natürlich auch die TUN/TAP und SKAS Unterstützung.

3.1.3 Kompilieren

Nachdem man seine Konfiguration gespeichert hat, landet man wieder auf der Kommandozeile. Ein abschließendes *make all* bzw. für den User-Mode-Kernel *make all ARCH=um* startet den Kompiliervorgang und nach zwei, drei, oder mehr Tassen Kaffee später hat man in der Datei *vmlinux* bzw. *linux* seinen neuen Kernel. Die eventuell als Modul aktivierten Funktionalitäten findet man im Unterordner *./lib* als so genannte Kernelmodule.

Nach den meisten Anleitungen folgt nun noch ein *make install*, auf das man gerade beim User-Mode-Kernel verzichten sollte. In der Regel sind die Funktionen, die die großen Distributionen zum hinzufügen eines neuen Kernels anbieten, nicht nur deutlich bedienerfreundlicher sondern verhindern auch ein anschließend unbenutzbares System. Des weiteren sind für den Aufruf *make install* root-Rechte erforderlich und für eine User-Mode-Umgebung kopiert man sich die genannten Dateien nun an die Stelle, von der man sie aus später starten möchte. Ein abschließendes *exit* beendet die Kommandokonsole wieder.

4 Das eigene Dateisystem

Nachdem man sich seinen Linux-Kernel an die Testumgebung angepasst hat möchte man sicherlich auch sein virtuelles System selber zusammenstellen. Dabei ist eigentlich gemeint sich ein eigenes root-Filesystem zusammenzustellen. So lange man dieses nur konfiguriert, d.h. mit Dateien füllt und Dateien innerhalb verändert, kann es logischerweise irgendwo im Dateibaum des Hostsystems eingehängt sein. Um Konfigurationsprogrammen ein Dateisystem, dass an der Wurzel eingehängt ist gibt es das Kommando *chroot*. Jedoch wird nicht nur den Konfigurationsprogrammen dieser Umstand vorgetäuscht, sondern auch dem Benutzer, dem bis zu einem *exit* jeglicher Zugriff auf seinen restlichen Dateibaum unmöglich ist. Da dieser Ordner später wirklich die Wurzel des virtuellen Systems ist sollte man vor allem darauf achten, alle Spuren einer anderen Position im Dateibaum zu beseitigen. Dazu zählen zum Beispiel Verknüpfungen auf außerhalb liegende Dateien oder absolute Pfadangaben.

4.1 Software installieren

Hat man das eigene root-Filesystem zum Beispiel mit Hilfe des loop-Gerätes und einer Image Datei in seinen Dateibaum eingehängt und hängt in diesen dann Installationsverzeichnisse der aktuellen Distribution, kann man sogar die Distributionsfunktionen zum Installieren der gewünschten Programme benutzen. Die großen Distributionen bieten nämlich die Möglichkeit einer Installation in Unterordnern an. SUSE Linux, als Beispiel, bietet sogar innerhalb seiner Konfigurationsumgebung *yast* die Option der Erstellung einer User-Mode-Linux Umgebung.

4.2 Besonderheiten für VNUML

Zur Simulation von Netzwerkszenarien mit VNUML benötigt VNUML zum Ablauf seiner Simulationsscripte und zur Konfiguration der virtuellen Systeme einige festgelegte Strukturen innerhalb des root-Filesystems der User-Mode-Umgebung.

- Im Wurzelverzeichnis muss sich ein leerer Ordner mit dem Namen /opt befinden.
- Innerhalb des Ordners des Initialisierungsprogrammes *init* muss sich ein symbolischer Verweis auf die Datei */opt/umlboot* befinden.
- Um das Szenario ohne ständige Eingabe des Administrator-Passwortes der User-Mode-Umgebung ablaufen zu lassen arbeitet VNUML mit SSH-Schlüssel-Authentifizierung. Dazu ist der symbolische Verweis des Ordners */root/.ssh* auf das Verzeichnis */opt* nötig.
- Zum Einhängen der individuellen Einstellungen in das jeweilige virtuelle System ist in die Datei /etc/fstab die Zeile */dev/udb1 /opt ext2 defaults 0 0* einzutragen.
- Damit VNUML sich auf den User-Mode-Umgebungen anmelden kann, muss nicht nur der SSH-Server installiert, sondern auch vom init-Programm gestartet werden. Dabei sollte er mit der Option -u0 ausgeführt werden.
- Neben den Programmen die für spezielle Simulation benutzt werden müssen auch die Programme *ifconfig*, *route*, *echo*, *hostname*, *halt* installiert sein. Des weiteren muss die *PATH*-Variable so gesetzt sein, dass diese Programme ohne Pfadangabe ausführbar sind.

5 Literatur

[1] User-Mode-Linux Entwicklerseiten, November 2005
http://user-mode-linux.sourceforge.net/

[2] VNUML Entwickler Dokumentation, November 2005
http://jungla.dit.upm.es/~vnuml/doc/1.5/user/index.html

[3] Linux intern. Technik, Administration und Programmierung; M. Wielsch, J. Prahm, H.-G. Eßer; 1. Auflage, DATA Becker Düsseldorf 1999.

[4] Linux-Magazin 01/2001
http://www.linux-magazin.de/Artikel/ausgabe/2001/01/virtual/virtual.html

[5] Vortrag zu Betriebssysteme Vorlesung der HTW-Dresden 2001, November 2005
http://wwwbs.informatik.htw-dresden.de/svortrag/i01/Schanze/stud_vortrag_s7854/user-mode-linux.html

[6] Gentoo HowTo, November 2005
http://www.gentoo.org/doc/en/uml.xml