BERUFSAKADEMIE | MANNHEIM
university of cooperative education | staatliche studienakademie

Referat

Linux 2.6 Kernelarchitektur

erarbeitet von

Markus Löschinger

Markus Wilke

Seminar Informationstechnik

Prof. W. Tischhauser

Inhalt

erstellt von
Markus Löschinger
Markus Wilke

1.0 Die Geschichte von Unix

Ende der 60er Jahre wollte ein Konsortium aus MIT, GE, Bell Labs, Honeywell und IBM ein Betriebssystem Namens Multics schaffen. Es sollte auf der Programmiersprache PL/1 basieren. Da sich die beteiligten Organisationen nicht über den Umfang und die Anforderungen einigen konnten, zog sich AT&T, der Inhaber der Bell Laboratories, aus dem Projekt zurück.

Die Entwickler der Bell Labs wollten aber nicht aufgeben und so beschloss ein Team um die Dennis Ritchie und Ken Tompson ein Mehrbenutzersystem zu entwickeln, das es ihnen ermöglichen sollte, gemeinsam zu programmieren.

Man entwickelte also für eine 18bit DEC PDP-7 ein Dateisystem, auf dem man alle Dateien, die angeschlossene Hardware und die laufenden Prozesse abbildete. Dahinter steht der UNIX-Grundsatz: Alles ist eine Datei.

Das gesamte System ist in Assembler programmiert, da für die PDP-7 keine Entwicklungsumgebung zur Verfügung stand. Es wurden auch noch einige kleine Tools entwickelt, die das System benutzbar machten. Nachdem das System auf der PDP-7 lief, wurden Mittel zur Anschaffung einer PDP-11 genehmigt. Um das Betriebssystem einfach portieren zu können musste der Quellcode in einer höheren Programmiersprache vorliegen. Deshalb wurde die Programmiersprache B weiterentwickelt. So entstand C, die UNIX-Standardsprache. Der Quellcode bestand aus 10.000 Lines of Code, die in C geschrieben waren und 1000 Zeilen Maschinenabhängigem Assembler.

Kernighan, ein weiterer UNIX-Entwickler, nannte das System spöttisch Unics; „emasculated Multics is Unics“(Es konnten anfänglich nur 2 Benutzer gleichzeitig am System arbeiten). Dieser Name wurde später in Unix gekürzt und von Ritchie aus purer Begeisterung für Kapitälchen in UNIX geändert. Der Name UNIX ist kein Akronym und hat auch sonst keine tiefsinnige Bedeutung. UNIX ist eingetragenes Warenzeichen von AT&T.

Da AT&T zur damaligen Zeit keine neuen Märkte erschließen durfte entschied man sich UNIX Version 6 zum Preis der Datenträger an Universitäten zu verteilen. Die Universität von Berkley engagierte sich sehr und wurde nicht zuletzt durch die Gastproffesur Thompsons zu einem Zentrum der UNIX-Weiterentwicklung. Hier wurde unter anderem TCP/IP in UNIX eingebunden. Es entstand sogar eine eigene Betriebssystemdistribution, die BSD (Berkley System Distribution) genannt wurde.

Anfang der Achtziger betrat AT&T offiziell den Computermarkt und begann UNIX System V kommerziell zu vermarkten. Auch die Universität von Berkley verkaufte ihr 4.2BSD. Als man merkte, dass sich die 2 UMIX-Varianten immer mehr von

einander entfernten, wurde das POSIX Standardisierungsgremium ins Leben gerufen. Dieser Standard sollte einen kleinsten gemeinsamen Nenner definieren, er wurde 1988 sogar IEEE-Standard.
Es bildeten sich noch weitere Firmenzusammenschlüsse, die jeweils einen eigenen Standard gründeten. Hier sind die Open Software Foundation, die aus den Firmen DEC, Siemens, HP und IBM bestand, die Unix International(Olivetti, Unisys, AT&T, Sun) und X/Open, die von Europäischen Unternehmen gegründet wurde, um die eigenen Interessen besser gegen die US-Firmen vertreten zu können.
UNIX System V Release 4 stellt einen Standard dar, der alle Verbesserungen aus BSD im AT&T UNIX einführt. Er wurde 1989 veröffentlicht und gilt heute noch als Referenz. Sun Solaris beispielsweise basiert auf diesem Standard.

Die aktuellen Rechtsstreitigkeiten zwischen der SCO Group und IBM basieren auf der Tatsache, das SCO die Rechte an UNIX erworben hat und nun behauptet, das IBM UNIX-Quellcode in den Linuxkernel kopiert hat. Die Unix System Laboratories, die das Copyright am UNIX-Code besitzen, wurden von AT&T an Novell verkauft. 1995 verkaufte Novell UNIX an die unter anderem von Microsoft mitfinanzierte SCO. SCO war das bekannteste UNIX, das auf x86-Prozesssoren lief. Der weitgehend unbekannte Linuxdistributor Caldera übernahm SCO, änderte aber kurze Zeit darauf seinen Namen wieder zurüch in SCO.

2.0 UNIX

UNIX besteht aus einem Kernel, der allein Zugriff auf die Geräte hat und Prozesse verwaltet. Der Kernel stellt das Dateisystem zur Verfügung, das neben Systemaufrufen die wesentliche Schnittstelle für die Prozesse in den Kernel darstellt. Eine Vielzahl von Programmen inklusive eines C-Entwicklungssystems und eines Textsatzprogrammes (troff) vervollständigen das System.

Das Dateisystem ist als hierarchisches Verzeichnis mit beliebigen Unterverzeichnissen organisiert, ein damals neues Konzept, das heute überall selbstverständlich ist. Wurzelverzeichnis (Root-Verzeichnis) dieser Hierarchie ist das Verzeichnis "/". Eins der hervorragenden Grundkonzepte von UNIX ist, auch Disketten- und CD-Laufwerke, weitere Festplatten des eigenen oder fremder Rechner, Terminals, Bandgeräte und andere "special files" im Dateisystem abzubilden. Dieser verallgemeinerte Dateibegriff gehört zum Wesen von UNIX und ermöglicht eine einfache, einheitliche Schnittstelle für die verschiedensten Anwendungen. In manchen UNIX-Derivaten werden selbst Prozesse und deren Eigenschaften auf Dateien abgebildet (proc-Filesystem).

Der Kommandointerpreter, die Shell, – unter Unix ein normaler Prozess ohne Privilegien – sowie die Programmierbibliotheken ermöglichen dem Anwender eine unerreicht einfache Ein-/Ausgabeumleitung in diese Dateien, und über Pipes die Verkettung von mehreren Programmen. Eine große Sammlung von

einfachen Programmen kann so mit Hilfe der Programmiermöglichkeiten des Kommandointerpreters kombiniert werden und komplizierte Aufgaben übernehmen, der so genannte UNIX-Werkzeugkasten entsteht und löst das frühere Konzept der leistungsfähigen, allein stehenden, allumfassenden, unflexiblen Anwendungen ab.

3.0 Die Geschichte von Linux

3.1 Linus Benedict Torvalds

Linus Benedict Torvalds wurde am 28.12.1969 in Helsinki, Finnland geboren und ist Gründer und Koordinator des Softwareprojekts Linux.
Er studierte an der Universität von Helsinki Informatik. In diesem Rahmen programmierte er an einem Terminalemulator, der auf einem 386er PC unter dem Betriebssystem Minix lief. Linus wollte sich komfortabel auf den UNIX-Servern der Uni einwählen. Als er in den Emulator noch Dateisystemzugriffe einbauen wollte, entstand der Plan ein richtiges Betriebssystem zu programmieren. Dieses Vorhaben kündigte er am 25.08.1991 in der Newsgroup `comp.os.minix` an.
Am 17.09.1991 stellte er den Kernel mit der Version 0.01 öffentlich zum Download bereit. Diesem Kernel konnte man mit GNU-Tools erweitern und erhielt so ein komplettes UNIX-artiges Betriebssystem.
Nachdem Torvalds einige Jahre beim amerikanischen Chiphersteller Transmeta gearbeitet hatte kümmert er sich jetzt im Open Source Development Lab (OSDL) hauptberuflich um die weiterentwicklung des Linuxkernels.
Des Weiteren gehören ihm die Namensrechte am Linux.

3.2 Linux

Torvalds wollte sein Betriebssystem ursprünglich Freax nennen, aber da er mit dem Benutzernamen Linux arbeitete und dem FTP-Admin der Universität Helsinki dieser Name besser gefiel, wurde das Software-Projekt unter dem Namen Linux veröffentlicht. Dieser Name steht heute weltweit für das freie Betriebssystem.
Der Name Linux bezeichnet eigentlich nur den Kernel, da die Programme, die um den Kernel gelagert sind(Shell, Dateisystem, Tools) aus dem Gnu is not UNIX (GNU) Projekt stammen, bezeichnet man das komplette Betriebssystem als GNU/Linux.

Anfänglich stand der Linuxkernel unter einer Lizenz, die jede kommerzielle Nutzung verbot, diese Lizenz wurde aber durch der GNU-GPL (General Public License) ersetzt. Diese Lizenz erlaubt die Weiterentwicklung und Vermarktung

des Quellcodes unter der Bedingung, dass alle Weiterentwicklungen wieder unter der GPL stehen und der Quellcode immer mitgeliefert wird.

Der Kernel wird von Entwicklern rund um den Globus ständig weiterentwickelt. Es wurden bis heute folgende Major-Releases veröffentlicht:

17. September	1991 Initial Public Release	v0.01
5. Oktober	1991 Erster "offizieller" Kernel	v0.02
16. April	1994	v1.0
9. Juni	1996	v2.0
21. Januar	1999	v2.2.0
4. Januar	2001	v2.4.0
18. Dezember	2003	v2.6.0

Die erste Version, die als Fehlerfrei galt war v1.0. Die Versionsnummer kennzeichnet den Kernel, man unterscheidet „stabile“ Versionen von „Entwicklerversionen“. Ob ein Kernel als stabil gilt, erkennt man an der 2. Stelle der Versionsnummer. Entwicklerkernel tragen hier eine ungerade Nummer, beispielsweise 2.1.x.
Die zurzeit aktuelle Version 2.6.5 ist aus dem Entwicklerkernel 2.5.x hervorgegangen. Die Freigabe neuer Releases übernimmt Torvalds selbst. Wird ein neuer Entwicklerkernel gestartet, werden alle Verbesserungen und Vorschläge gesammelt. Sobald ein so genanntes „Featurefreeze“ stattgefunden hat, werden nur noch Verbesserungen zu aktuellen Funktionen angenommen. Neue Ideen werden nicht mehr verfolgt und nicht in den Kernel eingearbeitet. Ein weiterer Fixpunkt ist das „Codefreeze“. Nach diesem Zeitpunkt wird nur noch der aktuelle Code von Fehlern befreit. Ist diese Phase beendet, wird eine stabile Version veröffentlicht.
Zum heutigen Zeitpunkt gibt es keine aktuelle Entwicklerversion, da der Kernel 2.6.x erst vor 5 Monaten veröffentlicht wurde und noch intensiv weiterentwickelt wird.
Torvalds kümmert sich nicht allein um die Entwicklung des Kernels, die Pflege der stabilen Versionen wird an andere Entwickler übertragen. Heute arbeiten folgende Programmierer als „Projektleiter“:

David Weinehall	2.0.x
Marc-Christian Petersen	2.2.x
Marcelo Tosatti	2.4.x
Andrew Morton	2.6.x

Linux läuft auf fast allem, was Einsen von Nullen unterscheiden kann. Angefangen von Miele Waschmaschinen über Playstations und PCs bis hin zu HighEnd Servern. Dies ist der Tatsache zu verdanken, das der Quellcode in C

geschrieben ist und frei verfügbar ist. Nur NetBSD läuft auf annähernd genauso vielen Hardwarearchitekturen.
Eine Linuxdistribution besteht aus einer Vielzahl von Programmen und Tools, die das Betriebssystem erweitern und das Arbeiten angenehmer machen sollen. Einige Distributionen verwenden jedoch keine Originalkernel, sondern erweitern sie, um sich einen Wettbewerbsvorteil zu verschaffen. So sind die Kernel von SuSE und RedHat nicht immer zu offiziellen Patches kompatibel. Hier muss der Anbieter eigene Updates entwickeln.

4.0 Detaillierte Vorstellung diverser (Weiter-)entwicklungen

Auszug der Neuimplementierungen und Weiterentwicklungen auf einen Blick:

- ACPI
- AGP 3.0
- AIO
- ALSA
- Bluetooth
- Build-System
- CAPI- Treiber
- CIFS
- DVB- Treiber
- Filesysteme
- Futexes
- IPSec
- ISDN
- LSM
- LVM2
- Module Loader
- NFS4
- NPTL
- NUMA
- PCMCIA
- PnPBIOS
- PowerNow
- Preemption
- rmap- VM
- SATA
- Scheduler
- Speedstep
- Tagged Command Queueing
- TCP/IP-Stack
- USB 2.0

Um den Umfang dieser Arbeit nicht zu sprengen wird nur auf einen Teil der hier erwähnten Neuerungen eingegangen. Diese werden zum Teil kurz angerissen oder in einem eigenen Kapitel detailliert erläutert.

4.1 Es geht doch noch nicht

Wie bei jeder Neuerung hat auch Linux mit Kinderkrankheiten zu kämpfen, so funktionieren derzeit die hptraid/promis- Treiber nicht. Des Weiteren gibt es Treiber, die sich nicht kompilieren lassen, da sie auf eine API Spezifikation zu warten scheinen. Einige Dateisysteme laufen auch noch nicht so stabil, wie man es von Linux gewohnt ist. Und RadHat 9 Nutzer können keine rpm- Pakete nutzen.

4.2 Da fehlt doch was

Vom 2.4er Kernel muss noch HFSPlus und SuperH 64 portiert werden.

4.3 Abgespeckt

Um den Kernel schlank zu halten und überschaubar, wurden einige Funktionen entfernt, so zum Beispiel khttpd oder alte Direct Rendering Mangager (ältere XFree86, als Version 4.1.0 laufen nicht). Booten von Diskette und der Systemaufruf table werden nicht länger unterstützt. Ebenso wurde LVM1 aus dem Kernel genommen.

5.0 Block-I/O-Layer

Lese- und Schreiboperationen auf Massenspeichern sind effizienter und schneller geworden. Es sind viele Limitierungen aufegehoben worden. Auf 32-Bit-Architekturen sind Blockdevices bis zu 16 TByte, auf 64-Bit-Architekturen bis zu 8 ExaByte möglich. Dies bietet ausreichend Platz für große RAID-Arrays, die durch den Logical Volume Manager LVM2 möglich sind. Eine Reihe von Aufgaben wie das Tagged Command Queuing, die bislang jeder Gerätetreiber selbst erledigen musste, sind in das Block-I/O-System gewandert. Das macht die Treiberprogrammierung einfacher und weniger fehleranfällig. Daten, die ein Prozess auf einen Massenspeicher schreibt, nimmt das virtuelle Filesystem (VFS) im Linux-Kernel entgegen und legt sie im Speicher ab.
An dieser Stelle kommt das neue I/O-Subsystem ins Spiel. Es arbeitet die zu schreibenden oder auch gelesenen Daten nicht mehr in 4-KByte-Blöcken ab, sondern kann beliebig große Datenblöcke an einem Stück zur Hardware schicken. Hierdurch kann die CPU-Last enorm reduziert werden. Ein I/O-Scheduler optimiert dabei den Plattenzugriff und sorgt dafür, dass kein Prozess beim Warten auf I/O verhungert und die Latenzen vor allem beim Lesen gering bleiben.

Da der I/O-Scheduler jetzt als eigenes Modul im Block-I/O-System realisiert ist, stehen im Kernel 2.6 verschiedene I/O-Scheduler zur Verfügung. In den meisten Situationen garantiert der antizipatorische I/O-Scheduler optimale Performance, dieser ist als default gesetzt. Lediglich in Umgebungen, in denen seek-Operationen dominieren, zum Beispiel in einer Datenbank, kann der einfachere Deadline-I/O-Scheduler eine 10 bis 20 Prozent höhere Leistung bringen, da er keine Zeit mit dem Versuch verschwendet, die zufällig verteilten Lese- und Schreiboperationen vorherzusagen.

6.0 Scheduler

In Kernel 2.6 wurde auch Ingo Molnars O(1)-Scheduler integriert Der alte Linux-Scheduler muss bei jedem Task-Switch die Liste der laufbereiten Prozesse durchsuchen und dann entscheiden, welcher Prozess an der Reihe ist. Das kostet umso mehr Zeit, je mehr Prozesse auf ihre Ausführung warten. Das System wird immer träger, wenn mehr Prozesse laufen. Der neue Scheduler braucht zum Taskwechsel immer gleich lang, unabhängig von der Anzahl der Prozesse: Der Kernel skaliert besser mit steigender Prozesszahl, interaktive Prozesse reagieren unter hoher Last schneller auf Benutzeraktionen. Die Grundidee des O(1)-Schedulers besteht darin, die laufbereiten Tasks, wo bei es egal ist, ob es sich um einen Prozesse oder Thread handelt, in ein Array aus 140 verketteten Listen zu stecken. Diese Queues 0 bis 99 für Prozesse mit Echtzeitpriorität, die restlichen 40 für die nice-Werte von -20 bis 19. In einer Bitmap zeigen 140 Bits an, welche Listen laufbereite Tasks enthalten. Um herauszufinden, welche Task als nächstes dran ist, muss der Scheduler lediglich das erste gesetzte Bit in der Bitmap finden. Damit niedrig priorisierte Tasks nicht verhungern, arbeitet der Scheduler mit zwei Arrays. Sobald eine Task im aktiven Array alle Zeitscheiben aufgebraucht hat, die ihr zustehen, wird sie in ein Array mit abgelaufenen Tasks verschoben. Wenn das Array mit den aktiven Tasks leer ist, tauscht der Scheduler die beiden Arrays aus und beginnt von vorne. In SMP-Systemen führt der Kernel für jede CPU eine eigene solche Runqueue durch. Ein Load-Balancer im Scheduler überprüft regelmäßig, ob die Runqueues gleichmäßig mit Prozessen bestückt sind. Sobald die Auslastung der Prozessoren um mehr als 25 Prozent voneinander abweicht, werden Prozesse verlagert, um ein Gleichgewicht herzustellen. Dabei unterscheidet der Scheduler zwischen physikalisch verschiedenen Prozessoren und den lediglich logischen CPUs von Hyperthreading-Prozessoren. Ebenfalls der gefühlten Systemgeschwindigkeit zugute kommt der unterbrechbare Kernelcode (preemptible kernel). Im Kernel 2.4 schläft der Task-Scheduler, solange der Prozessor Kernelcode ausführt. Ab 2.6 sind auch Kernelfunktionen an vielen Stellen unterbrechbar, nur noch wenige Codeteile des Kernels sind weiterhin durch Spinlocks geschützt. Das low latency Projekt hat zudem Patches beigesteuert, die längere kritische Codeteile im Kernel aufteilen und mit preemption points versehen Diese Änderungen vermindern die Latenzen bei Interrupts und Taskwechseln dramatisch (Vergleichsmessungen zufolge um Faktor 10 und mehr), sodass Linux schon in den Bereich der weichen Echtzeitanwendungen vorstößt, wobei es sich um keine absolute Echtzeitfähigkeit handelt. Ein weiteres viel besprochenes neues Merkmal ist der von Ingo Molnar überarbeitete Prozessscheduler, der einen O(1)-Algorithmus nutzt. Bei niedrigen Lasten sollte man keine Veränderungen feststellen. Bei einer hohen Zahl an Prozessen jedoch, speziell bei großen SMP-Systemen, sollte es eine deutlich verbesserte Skalierbarkeit geben.

7.0 asynchrones Block-IO

In 2.6 wurde Unterstützung für Kernel AIO eingebunden. Das AIO aktiviert einen einzelnen Anwendungsthread I/O-Prozesse, und kann damit andere Prozesse überlappen. Dies geschieht, indem eine Schnittstelle angeboten wird, wo ein oder mehrere I/O-Abfragen in einen Systemaufruf, einen `io_submit`, getätigt werden. Dabei wird nicht auf die Beendigung dieses Aufrufes gewartet. Es wird zusätzlich eine separate Schnittstelle, `io_getevents`, aufgerufen, um von beendeten I/O Operationen das Ergebnis zu bekommen. Diese werden dann mit einer gegeben Komplettierungsgruppe assoziiert.

8.0 Futexes

Rusty Russell hat Futexe (Fast Userspace Mutexes - Schnelle Benutzerebene Mutexe) implementiert, diese machen die Synchronisation von Threads effizienter. Dies geschieht durch die Möglichkeit Threads schlafen zu schicken, wenn sie darauf warten, dass sich ein auf Lock gesetzter Speicherwert ändert. Über einen Systemcall kann dieser Thread dann solange schlafen, bis sich der Speicherwert ändert. Dies erspart die regelmäßigen Abfragen über aufwendige Signalkommunikation.
Futexes sind nicht nur im Bezug auf Threads ein Zeitersparnis, sondern werden als allgemein nutzbare Methode zur Synchronisation von Userspache Prozessen mit Kernelhilfe angesehen.

9.0 Gerätemodell

Eine weitere Neuerung des Kernels ist ein vereinheitlichtes Gerätemodell. Alle Geräte sind im Kernel in einer Liste von kobject-Strukturen repräsentiert, die über ein neues Pseudodateisystem, das system filesystem (sysfs), dem Anwender zugänglich gemacht wird.
Dieses neue Gerätemodell unterscheidet Busse, Geräte und Klassen. Die Beziehungen zwischen Bussen und Geräten sind die selben, wie die physischen Gegebenheiten im Rechner, wobei auch virtuelle Geräte möglich sind. Die Geräte werden in Klassen nach althergebrachten Schemata wie Eingabegeräte, Netzwerk oder TTYs, geordnet. Diese Unterscheidung zeigt sich auch im Aufbau des sysfs, in welches die meisten Informationen aus `/proc` portiert werden sollen: `/proc` soll mittelfristig wieder auf Prozessinformationen beschränkt werden.
Das neue Gerätemodell verlagert also das Wissen über die Hardware in den Kernel. Das verbessert nicht nur Powermanagement und Systemkonfiguration via ACPI, sondern macht es auch möglich, alle Geräte als hot-pluggable zu

behandeln: Aus Kernelsicht besteht dann kein prinzipieller Unterschied zwischen einer PCI-Karte und einem USB-Gerät. Selbst ein Prozessor ist im laufenden Betrieb ausbaubar (hierfür gibt es einen Patch). Insgesamt dürfte das neue Gerätemodell den Umgang mit Hotplug-Geräten aller Art deutlich vereinfachen und verschnellern.

10.0 rmap- VM

In die 2.6er-VM ist der rmap-Patch die Rückwärtsabbildungen ('reverse mapping') von Rik van Riel integriert. Speicherseiten wurden bis Version 2.4 des Linux-Kernels in einer Datenstruktur verwaltet, die es nur erlaubte, die physische Speicherseite zu einer logischen Adresse zu finden. Der 2.6er Kernel kann in einer verketteten Liste von Page Table Entries (PTE-Liste) zu jeder physischen Speicherseite nachschlagen, um herauszufinden welche Prozesse sie referenzieren. Auf diese Weise wird die Suche nach nicht mehr verwendeten Speicherseiten beschleunigt, die dann freigegeben und anderen Prozessen zur Verfügung gestellt werden können. Es werden nicht mehr alle virtuellen Speicheradressen durchsucht, sondern die rmap-VM kann die physischen Seitenadressen auf "freizugebende" Seiten scannen. Das bringt nicht nur bei Speicherknappheit, sondern auch bei viel physischem Speicher Vorteile.

Die VM prüft Sparse-Swapdateien und bricht ab, wenn sie eine findet. Zu nennen ist, dass Verson0-Swap- Partitionen nicht mehr unterstützt werden. /proc/sys/vm/swapponess definiert die Präferenzen ders Kernel für den Pagecache und Bildspeicher (mapped memory). Wenn dieser Wert auf 100 (%) gesetzt eingestellt wird, behandelt der Kernel beide Speichertypen gleich. Wenn es jedoch auf 0 (%) eingestellt wird, macht der Kernel eine Rückforderung von Pagecache und nicht von Speicher, welcher in Pagetables abgelegt wäre. Die Anzahl der Swapdateien wurde von 64 auf 32 heruntergesetzt die immer noch 64GB groß sein können aber in der Regel die 2GB nicht überschreiten, da sonst zusätzliche Tools nötig wären.
Durch diese diversen Veränderungen sollten Swapdateien genauso schnell wie Swappartitionen sein.

11.0 Preemption

Preemptions- Patches wurden viel diskutiert, sind aber in den Kernel 2.6 mit eingeflossen und sollten die Latenzen bei multimedialen Anwendungen erheblich verbessern. Die Preemption muss allerdings abgeschaltet werden, wenn Pro-CPU- Daten genutzt werden.
Die Meldung xxx exited with preempt count=n im Syslog bedeutet, dass gesperrt wurde und diese Sperre nicht wieder aufgehoben wurde. Dies ist aber kein schwerwiegender Fehler.

12.0 Thread-Modell (NPTL)

Bei den älteren Thread- Methoden war das Hauptproblem die Skalierbarkeit, Native POSIX Threading Library (NPTL) und Non-Uniform Memory Architecture (NUMA) bringen großen Unternehmensanwendungen (Servern) eine leistungsfähigere Threadimplementierung. Bisher ließen sich auf einer x86 Architektur 8192 Threads erzeugen. NPTL-Threads haben keine feste Obergrenze und können, unabhängig wie viele Threads schon laufen, mit einem einzelnen Aufruf gestartet werden. Die große Anzahl von Threads ist dank eindeutigem PID- Hash kein Problem oder eine Verlangsamerung. Nutzer sollten eine signifikante Beschleunigung in grundlegenden Threadoperationen feststellen.
Im Bereich des Signalhandlings und verwandten Threading-Semantiken plus Coredumping können hier Fehler auftreten, die noch bereinigt werden müssen.

13.0 Filesysteme

Hier eine Liste der im Kernel 2.6 unterstützten Filesysteme:

- adfs
- afs
- amiga ffs
- apple macintosh BeOS befs (ro)
- bfs
- cramfs
- efs (ro)
- ext2
- ext3
- free vxfs
- ifs
- iso9660
- minix
- msdos
- ntfs (ro)
- os/2 hpfs
- qnx4fs
- reiserfs4
- romfs
- sysvfs
- udf
- ufs
- vfat
- xfs

Der neue NTFS-Treiber gilt als stabiler und schneller, dabei werden mehr NTFS-Features unterstützt und es sind erstmals sichere Schreiboperationen auf NTFS-Partitionen erlaubt.

Ext2 und ext3 kennen jetzt erweiterte Attribute, sowie Access Control Lists (ACLs) nach POSIX-Standard. Der HTree-Patch beschleunigt Verzeichnisoperationen in „beladenen“ Directories, dies war bislang eine Schwäche von ext2/ext3.

14.0 Zusammenfassung

Dank schlankerem Code scheint der Ressourcenbedarf bei all dem nicht gestiegen zu sein, im Gegenteil: Unter hoher Last reagiert das System mit dem neuen Kernel flüssiger.

In einem Interview hat Linus Torvalds gesagt, für ihn sei die wichtigste Änderung im Kernel 2.6 das große Aufräumen in den Quelltexten, was die Wartbarkeit des Codes drastisch verbessert hat und Linux Zukunftssicherheit vorantreibt.

15.0 Quellen

- c´t 24/2003, „The Next Generation“,
  Dr. O. Diedrich
- Heise Newsticker (Meldung 45420)
- http://www.linux.org.uk/~davej/docs/post-halloween-2.6.txt - The post-helloween document
- http://www.wikipedia.org
- http://www.kernel.org