Das Betriebssystem GNU/Hurd

Auf neuen Wegen

Marcus Brinkmann

GNU/Hurd ist ein POSIX-kompatibles System, bietet darüber hinaus aber eine flexiblere Architektur und weitreichende Verallgemeinerungen. Das Ergebnis ist eine unbegrenzt erweiterbare Umgebung, die die Schranken zwischen dem Benutzer und dem System weitgehend aufhebt.

Hurd ist der von Thomas Bushnell BSG entworfene Betriebssystemkern des GNU-Systems und ersetzt zusammen mit einem Microkernel und der GNU-C-Bibliothek den klassischen monolithischen Unixkernel. Während bei jenen alle wichtigen Systemaufgaben und Hardwareschnittstellen in einem großen Programm, dem Kernel, vereint sind, finden sich in einem Microkernel nur primitive Operationen wie zum Beispiel der Austausch von Nachrichten (message passing), die Verwaltung der Rechenzeit (scheduling), die Durchsetzung der Speicherverwaltung (paging) und einfache Hardwareabstraktionen. Der Großteil des Systems wird von der C-Bibliothek und den Hurd Servern implementiert, die als normale Tasks ohne Kernelprivilegien laufen. Typische Aufgaben der Hurd Server sind zum Beispiel Socketimplementierungen, Prozeßverwaltung, Authentifikation von Benutzern und die Implementierung von Dateisystemen.

Der Vorteil im Vergleich zu monolithischen Systemen wie Linux liegt in der weitreichenden Modularität der Systemkomponenten. Damit sind nicht Kernelmodule wie bei Linux gemeint, die zur Laufzeit in den Kernel eingebunden werden, sondern echte Module, die in keiner Weise Bestandteil des Kernels sind. Dadurch erhöht sich die Stabilität des Gesamtsystems, denn eine einzelne Komponente, wie zum Beispiel ein Dateisystemserver, kann abstürzen, ohne daß andere Prozesse in Mitleidenschaft gezogen werden. Es folgt, daß auch nicht-privilegierte Benutzer eigene Dateisysteme in das System einbinden dürfen. Dem Programmcode des Servers muß nicht vertraut werden, da er keine speziellen Privilegien erhält.

Ein Nachteil ist, daß die Kommunikation zwischen den einzelnen Systemkomponenten erschwert wird, da nicht zwei Bestandteile auf gemeinsame Datensegmente zugreifen können. Die nunmehr absolut erforderliche Interprozeßkommunikation (IPC) gestaltet sich aufwendiger, insbesondere da man eine weitgehende Transparenz erreichen möchte. Dies ist ganz besonders wichtig für verteilte Systeme, in denen ein Server auch auf einem anderen Rechner im Netzwerk ausgeführt werden kann. Diese Transparenz erreicht man durch entfernte Prozeduraufrufe (Remote Procedure Calls, RPC), die wie normale Funktionsaufrufe in einem Programm benutzt werden können, in Wirklichkeit aber eine mitunter komplexe Kommunikation mit dem Server durchführen. Die Argumente werden dabei als Datensätze übermittelt und das kalkulierte Ergebnis zurückgeliefert. Die entstehenden Mehrkosten senken die Ausführungsgeschwindigkeit der Programme, aber grundsätzlich ist anzumerken, daß der Gesamtmehraufwand sehr klein gehalten werden kann. Zudem ist der Aufwand konstant, was zur Folge hat, daß bei schnelleren Rechnern die zur Interprozeßkommunikation aufgewandte Rechenzeit weniger ins Gewicht fällt. In der momentanen Implementierung ist allerdings noch keine Optimierung vorgenommen worden.

GNU Mach

Als zugrundeliegender Kernel wird vom GNU/Hurd Projekt ein Abkömmling des Mach-Microkernels verwendet. Dieser ist zwar ein wenig in die Jahre gekommen, stellt dafür aber eine reichhaltige Semantik zur Verfügung und läuft sehr stabil. Zuletzt von der Universität von Utah entwickelt, wurde der Kernel von den Hurd-Entwicklern modifiziert und als GNU-Mach herausgegeben. Neben dem Nachrichtenaustausch, dem Scheduling und der Verwaltung von Tasks und Threads stellt Mach auch eine Hardwareschnittstelle zur Verfügung. Dahinter wurden die blockorientierten Geräte und Netzwerkkartentreiber des Linux-2.0-Kernels eingearbeitet, so daß wir eine breite Palette an solchen Geräten unterstützen können. Auch eine Einbindung der zeichenorientierten Geräte ist möglich und weitgehend entwickelt, aber hier steht noch die veraltete Terminalschnittstelle von Mach im Weg. Die Zukunft liegt jedoch mit Sicherheit in der Auslagerung der Gerätetreiber aus Mach. Was hier möglich ist, hat Roland McGrath gezeigt, einer der Hauptentwickler von Hurd.

Er hat Mach nach OSKit portiert, ein Sammlung von Hardwaretreibern von Linux und BSD.

Es gibt begründete Kritik an der Verwendung von Mach, aber diese trifft nicht schwer, denn die Hurd-Server sind weitestgehend unabhängig von der speziellen Implementierung des Microkernels entwickelt. Eine Portierung von Hurd auf einen modernen Microkernel wäre mit Sicherheit ein interessantes und nicht allzu schwieriges Unterfangen, Bestrebungen in diese Richtung stehen allerdings noch aus.

Hurd

Das Softwarepaket Hurd besteht aus einer großen Anzahl von Servern und einer Reihe von Hilfsprogrammen. Da das Hurd-System POSIX-kompatibel ist und somit eine wohldefinierte Programmierschnittstelle zur Verfügung stellt, läuft eine Vielzahl von Programmen ohne oder nur mit geringen Veränderungen. Deswegen sind im Hurd-Quelltext nur systemabhängige Werkzeuge enthalten, wie zum Beispiel »ps« (process status) oder »vmstat« (virtual memory status). Die weiteren Betriebssystemkomponenten finden sich übrigens in den weitläufig bekannten GNU-Paketen »shellutils«, »fileutils« und anderen.

Die Hurd-Server kooperieren stark mit der GNU-C-Bibliothek, so zum Beispiel in der Verarbeitung von Signalen, aber auch in der Rechtevergabe und Prozeßverwaltung und vielen weiteren Bereichen. Dabei verwenden beide Mach-Systemaufrufe, unter anderem zur Kommunikation. POSIX-kompatible Programme nutzen in der Regel ausschließlich die GNU-LibC-Schnittstelle, aber ein Programm, das spezifische Dienste eines Hurd-Servers benötigt, die nicht über die C-Bibliothek zur Verfügung gestellt werden, können auch direkt mit den Hurd-Servern kommunizieren. Dazu stellt das Hurd-System entsprechende Bibliotheken zur Verfügung, die auch von den zum Hurd-Paket gehörenden Hilfsprogrammen verwendet werden. In Bild 1 sind diese Beziehungen schematisch dargestellt. Ein Pfeil von A nach B bedeutet hier, daß die Komponente A die Komponente B verwendet. Um einen besseren Eindruck von der Aufteilung des Systems und den Verantwortungsbereichen der einzelnen Server zu bekommen, sollen nun einige der wichtigsten Server des Hurd-Systems vorgestellt werden.

Der Ausführungsserver exec

Um ein Programm auszuführen, verwendet die C-Bibliothek den exec-Server. Dieser muß zum einen das Objektformat erkennen und zum anderen Statusinformationen verwalten, wie zum Beispiel die übergebenen Argumente, die Umgebungsvariablen und das aktuelle Arbeitsverzeichnis. Zum Erkennen des Objektdateiformats verwendet der exec-Server die BFD-Bibliothek (Binary File Deskriptor), die eine Vielzahl von Objektdateiformaten erkennt (unter anderem »aout« und »elf«). Der exec-Server behandelt auch Skripte mit der #!-Notation, indem der entsprechende Interpreter gestartet wird.

Zur Verdeutlichung sei noch einmal darauf hingewiesen, daß weder die C-Bibliothek noch der Mach-Microkernel, noch die anderen Hurd-Server etwas über Objektdateiformate wissen müssen. In einem monolithischen System wie zum Beispiel Linux kann nur der Administrator die Fähigkeit, bestimmte Objektdateiformate (wie »aout« oder »java«) zu erkennen und auszuführen, dem System hinzufügen, etwa durch Kompilation eines neuen Kernels oder das Hinzufügen eines Kernelmoduls. Allerdings wird dies selten der Fall sein, besonders wenn ein Benutzer ein experimentelles oder selbst entworfenes Objekdateiformat verwenden möchte. Das Risiko, absichtlich oder unabsichtlich die Systemsicherheit zu gefährden, wäre viel zu groß. In Hurd kann ein Benutzer über die Umgebungsvariable *EXECSERVERS* seinen eigenen exec-Server angeben, der ein beliebiges Objektdateiformat lesen und zur Ausführung bringen kann. Der Benutzer kann so das System selbständig erweitern und seinen Bedürfnissen anpassen. Da der neue *exec*-Server als normaler Prozeß mit den Rechten des Benutzers läuft, bleibt die Systemsicherheit gewahrt. Das Hurd-System ist so entworfen, daß auch unter Einbeziehung der komplexen Benutzerrechte (man denke nur an *setuid*-Programme) kein Benutzer durch eigene Server mehr Rechte erhalten kann als ihm der Administrator zugesteht.

Die hier dargelegte Flexibilität erstreckt sich auf das ganze Hurd-System. Fast überall wird es dem Benutzer ermöglicht, Systemkomponenten durch eigene Implementierungen zu überschreiben, ohne andere Benutzer zu beeinflussen oder die Gesamtsicherheit des Systems zu gefährden. Der Prozeß-Server »proc« erwei-

Wichtige Internetadressen

GNU/Hurd Homepage	*http://hurd.gnu.org/*
Debian GNU/Hurd	*http://www.debian.org/ports/hurd*
Installation	*http://www.debian.org/ports/hurd/hurd-install*
Source Code	*http://www.gnu.org/software/devel*
Binaries	*ftp://www.debian.org/debian/dists/unstable/main/binary-hurd-i386*
Diverses	*ftp://alpha.gnu.org/gnu/hurd*
Mailing-Liste	*debian-hurd@lists.debian.org*
OSKit	*http://www.cs.utah.edu/flux/oskit*

Textbox 1

tert die Mach Tasks zu vollständigen POSIX-Prozessen. Da Mach-Tasks nicht die komplexe Semantik von POSIX-Prozessen haben, ist es notwendig, die zusätzlichen Informationen irgendwo festzuhalten, was im proc-Server geschieht. Dies bezieht sich zum Beispiel auf Prozeßgruppen, zu einem Prozeß gehörende Terminals, Benutzerrechten des Prozesses und ähnlichem. Der proc-Server verwaltet somit eine Eins-zu-Eins-Beziehung zwischen Mach-Tasks und POSIX-Prozessen. Das Hurd-Programm »ps« zum Beispiel erhält die notwendigen Status Informationen vom proc-Server.

Der Authentifikationsserver auth

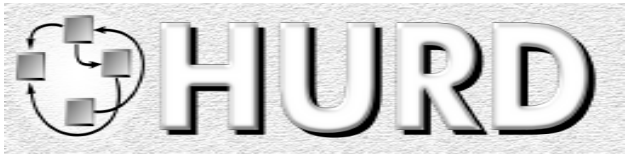
Der Authentifikationsserver »auth« übernimmt die Verwaltung der Benutzerrechte und hat somit eine zentrale Stellung im Hurd-System. Jede Verbindung (port) zu diesem Server ist gekennzeichnet durch eine Menge von Benutzerkennungen und Gruppenkennungen. Jede der beiden Mengen kann leer sein. Bestehende Verbindungen lassen sich zusammenlegen, aber nur eine Verbindung mit der Benutzerkennung 0 (root) kann neue Verbindungen mit beliebigen Kennungen erzeugen. Dadurch ist sichergestellt, daß kein nicht-privilegierter Benutzer ihm nicht zugeteilte Rechte erlangt.

Hier findet sich übrigens eine Erweiterung gegenüber POSIX. Ein Benutzer kann eine beliebige Anzahl von verfügbaren Benutzer- und Gruppenkennungen haben. Das Hurd-Hilfsprogramm »ids« zeigt die verfügbaren Kennungen, die Programme »setauth«, »rmauth« und »addauth« ermöglichen es, die Kennungen eines Benutzers zu editieren. Es ist auch möglich, überhaupt keine Kennungen zu besitzen: Dies ist bei einem Benutzer der Fall, der sich noch nicht eingeloggt hat. Das macht keinen Sinn? In einem traditionellem Unix-System sicher nicht. Aber das Standard Hurd-Login-Programm stellt anstatt eines einfachen login-Prompts eine Shell zur Verfügung. Um dies zu kontrollieren, haben alle Hurd Dateien auch noch einen vierten Satz von »rwx«-Zugriffsrechten auf Dateien und Verzeichnisse für diesen »Nicht-Benutzer«.

Der Absturzserver crash

Die wichtigste Funktion des »auth«-Servers ist es jedoch, zwischen einem Server und einem Client auf sichere Weise zu vermitteln, ohne daß einem der beiden Kommunikationspartner vertraut werden muß. Nur so kann das Sicherheitsschema auch wirklich systemweit durchgesetzt werden. Aufgrund dieser besonderen Stellung muß auch ein »auth«-Server für jeden Server besonders gekennzeichnet sein. Ein Benutzer kann also nicht den »auth«-Server für beliebige Server im System überschreiben, sondern höchstens für seine eigenen Server.

Bei einem Programmabsturz wird der »crash«-Server



aktiv. Der Default-crash-Server (auch hier kann ein Benutzer seine eigene Implementierung aktivieren) unterstützt momentan zwei Aktionen: Das Programm kann in den Hintergrund gestellt werden (suspend), um das Anfügen eines Debuggers zu ermöglichen, oder das Programm kann sofort aus dem Speicher entfernt werden (kill). Ein Speicherauszug (core dump) ist vorgesehen, aber noch nicht implementiert.

Übersetzer

Nun zu einem der wesentlichsten Konzepte des Hurd-Systems: Die Übersetzer (translator). Erst diese liefern den Benutzern die maximale Flexibilität und Erweiterbarkeit des Hurd-Systems. Ein Übersetzer ist ein Programm, das als Schnittstelle ein Dateisystem zur Verfügung stellt. Die Kommunikation zwischen dem Benutzer und dem Programm findet dabei durch die Verzeichnishierarchie und den darin enthaltenen Dateien statt. Als konkretes Beispiel sei hier als erstes der FTP-Übersetzer genannt, der einen transparenten Zugriff auf FTP-Server im Internet ermöglicht. Um das Beispiel zu konkretisieren, hier der Befehl, der eingegeben werden muß, um durch das Verzeichnis »~/gnu/« auf den GNU-FTP-Server zuzugreifen:

```
settrans ~/gnu /hurd/ftpfs ftp.gnu.org:/
```

Als Ergebnis finden sich die Dateien und Verzeichnisse des GNU-FTP-Servers im Benutzerverzeichnis »~/gnu/«. Das Verzeichnis wird von nun an mit der URL »« ftp://ftp.gnu.org/« identifiziert. Um die Dateien zu bearbeiten, kann der Benutzer nun die normalen Befehle anwenden, die ihm schon für lokale Dateien zur Verfügung stehen, zum Beispiel »cat«, »less«, »cp«, aber auch die automatische Pfadvervollständigung der Shell (tab-completion) funktioniert wie gewohnt. Keines dieser Programme muß speziell dafür umgerüstet werden: Der Netzwerkzugriff erfolgt vollkommen transparent durch den ftpfs-Server.

Wie funktioniert das? Der obige *settrans*-Befehl hat in der Inode »~/gnu« in einem speziell dafür vorgesehenen Feld den Verweis »/hurd/ftpfs ftp.gnu.org:/« geschrieben, der bei einem Zugriff auf das Verzeichnis »~/gnu« (genauer: die entsprechende Inode) vom übrigen Hurd-System ausgewertet wird. Dabei wird zunächst geprüft, ob der Übersetzer schon als Prozeß läuft. Ist dies nicht der Fall, wird er automatisch gestartet und ihm die bei *settrans* angegebenen Parameter übergeben, in diesem Fall »ftp.gnu.org:/« als einziges Argument. Dann wird der ftpfs-Übersetzer gefragt, was auf die

Anfrage des Benutzers geantwortet werden soll. Dabei ist im Prinzip alles möglich: Ob ein Übersetzer nur eine einzige Datei, ein Verzeichnis, oder, wie hier, einen ganzen Verzeichnisbaum zur Verfügung stellt, bleibt allein dem Programmierer überlassen.

So stellt zum Beispiel der Übersetzer »/hurd/hello« nur eine einzige Datei zur Verfügung, die einen wohlbekannten Gruß enthält. Der Übersetzer »/hurd/null« stellt auch eine Datei zur Verfügung, die sich so verhält wie die Gerätedatei »/dev/null« in üblichen Unix-Systemen. Damit kommen wir zu einem wichtigen Anwendungsbereich der Übersetzer: Alle Gerätedateien im /dev/-Verzeichnis sind im Hurd-System durch Übersetzer verwirklicht, und laufen somit als normale Prozesse im Benutzerspeicher und nicht als privilegierte Kerneltasks.

Auch Dateisysteme sind naheliegenderweise durch Übersetzer verwirklicht. Das übliche Einbinden (mounten) von Dateisystemen ist einfach nur ein spezieller Fall des allgemeinen Übersetzerkonzepts. Momentan sind die Dateisystemformate BSD Fast File System, Extended 2 und ISO 9660 implementiert, letzteres allerdings nur ohne Erweiterungen. Hier ein Beispiel, um eine Floppy mit einem ext2-Dateisystem an »/floppy« zu binden:

```
settrans /floppy /hurd/ext2fs /dev/fd0
```

Die Anwendungsmöglichkeiten sind enorm. Von klassischen Dateisystemen über transparentem FTP-Zugriff und der Verwirklichung von Gerätedateien, symbolischen und harten Verweisen (links), fifos und Terminals kann alles als (manchmal entartetes) Dateisystem interpretiert werden. Aber auch andere Übersetzer sind denkbar, wie zum Beispiel ein signature-Server, der bei jedem Auslesen eine andere Signatur für die E-Mail des Benutzers ausspuckt. Da jeder Benutzer Übersetzer selbst programmieren und diese in das System einbinden kann, sind der Phantasie des Programmierers keine Grenzen gesetzt. Als ein Beispiel sei noch der hostmux-Übersetzer genannt, der sinnvoll mit dem FTP-Übersetzer kombiniert werden kann. Er interpretiert das erste Verzeichnis als einen Hostnamen und leitet die Anfrage an einen anderen Übersetzer weiter. Konkret kann das zum Beispiel so aussehen:

```
settrans /ftp /hurd/hostmux /hurd/ftpfs ${host}:/
ls /ftp/ftp.gnu.org/
ls /ftp/ftp.debian.org/debian
```

In diesem Beispiel wird bei der ersten Anfrage die Zeichenkette »${host}« durch »ftp.gnu.org« ersetzt und der FTP-Übersetzer mit den gleichen Argumenten wie oben gestartet. Bei der zweiten Anfrage wird statt dessen »ftp.debian.org« ersetzt, und ein weiterer FTP-Übersetzer gestartet.

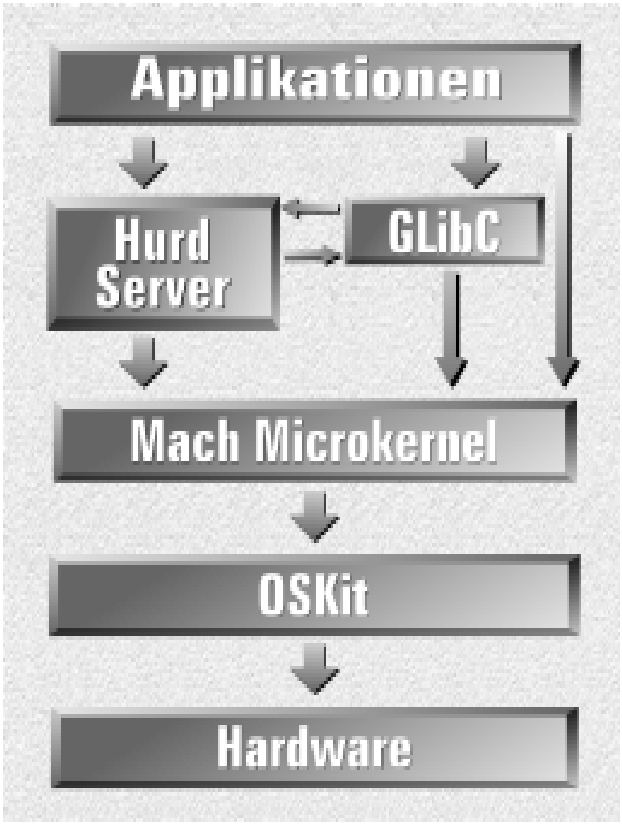


Bild 1. Abhängigkeiten zwischen den Systemkomponenten.

Es ist wichtig, zwischen dem passiven und dem aktiven Übersetzer zu unterscheiden: Mit dem ersten wird nur die in der Inode gespeicherte Zeichenkette bezeichnet, und diese bleibt auch beim Neustart des Rechners erhalten. Sie kann mit dem »showtrans«-Programm ausgelesen werden. Der aktive Übersetzer ist nicht immer vorhanden und bezeichnet den laufenden Prozeß. Beide sind im Prinzip unabhängig voneinander. Die beiden verbindet nur die Tatsache, daß der passive Übersetzer bei einem Zugriff auf die Inode ausgewertet wird, falls noch kein aktiver Übersetzer existiert. Um den Unterschied noch einmal zu verdeutlichen: Das exakte Äquivalent zu einem klassischen

```
mount -t ext2fs /dev/fd0 /floppy
```

unter Verwendung der Hurd-Übersetzer ist

```
settrans -a /floppy /hurd/ext2fs /dev/fd0
```

Entscheidend ist die Option »-a«, die angibt, den Übersetzer nicht als passiven Übersetzer in die Inode zu schreiben, sondern sofort zu starten. Nach einem Neustart muß dieser Vorgang dann bei Bedarf wiederholt werden. Im Gegensatz dazu erhält man bei Fortlassen der »-a«-Option eine Art »automounter«-Funktionalität.

Auch andere Systeme wie BSD und Linux können transparentes FTP und andere Dateisysteme anbieten, aber im Regelfall scheitert dies an der Aufgabenverteilung im monolithischen System: Diese Fähigkeiten gehören einfach nicht in den Kernel. Benutzerdateisysteme findet man im traditionellen Unix-Bereich nur in Ausnahmefällen, und die Unterstützung dafür muß im Kernel vom Administrator aktiviert werden. Im Hurd-System jedoch wurde diese Idee zum Prinzip erhoben und konsequent unter Berücksichtigung der Benutzerechte implementiert und eingesetzt.

Es sollte nicht unerwähnt bleiben, daß die Programmierung von Übersetzern durch Verwendung der Hurd-Bibliotheken erheblich vereinfacht wird, die einen Großteil der Arbeit erledigen. Im einfachsten Fall muß man »nur noch« die Platzhalterfunktionen der Dateizugriffsoperationen wie Lesen und Schreiben mit eigenen Implementierungen überschreiben. Für nicht-C-Programmierer arbeitet John Tobey an Perl-Bindings, um das Programmieren von Übersetzern in Perl zu ermöglichen.

Stores

Mit zum Hurd-System gehört auch die store-Bibliothek, welche einen Datenspeicher (store) abstrahiert und zur Verfügung stellt. Dies kann eine Mach-Gerätedatei (wie zum Beispiel eine Festplattenpartition) sein oder eine Datei in einem Dateisystem. Zudem stellt die store-Bibliothek Filter zur Verfügung, mit denen sich zum Beispiel einzelne Blocksegmente aus einem store auswählen und rekombinieren lassen. Auch eine automatische Dekompression (»gz« und »bz2«) ist möglich, sowie eine Verkettung von mehreren Datenspeichern. Die Filter lassen sich in beliebiger Reihenfolge hintereinanderschalten, auch mehrfach. Da die relevanten Hurd-Dateisysteme diese Bibliothek verwenden, können so zum Beispiel komprimierte Floppies ohne Aufwand eingebunden werden, in etwa so:

```
settrans /floppy /hurd/ext2fs -Tbunzip2:device /dev/fd0
```

Es gibt weitere Bibliotheken, die dem Programmierer die Arbeit mit Hurd-Servern oder Microkernel-Konzepten wie Ports und Threads innerhalb des Hurd-Systems erleichtern. Auch die Hurd-Server selbst machen von diesen Bibliotheken natürlich ausgiebig Gebrauch.

Hurd in Hurd

Neben der Erweiterbarkeit durch den Benutzer liefert die modulare Struktur des Hurd-Systems auch den Vorteil höherer Stabilität, denn wichtige Aufgaben werden nicht mehr vom privilegierten Kernel vollzogen, sondern von normalen Benutzerprozessen. Wenn diese abstürzen, kann der Rest des Systems weiter störungsfrei operieren. Davon ausgenommen sind nur die zentralen Server »exec«, »proc«, »auth und der Dateisystemserver des Wurzelverzeichnisses, denn ohne diese ist das System nicht vollständig lauffähig. Daraus folgt, daß die Entwicklung und Einbindung neuer Komponenten, beispielsweise neuer Dateisysteme, vollständig ohne spezielle Privilegien durch jeden Benutzer stattfinden kann, denn es muß zum Beispiel kein neuer Kernel kompiliert und durch Neustart des Rechners ausprobiert werden. Dies wird unterstützt durch die Möglichkeit, komplette Hurd-Systeme innerhalb eines Hurd-Systems hochzufahren. Dazu benötigt man nur ein eigenes Wurzeldateisystem. Mit dem Befehl »boot« kann dann ein nicht privilegierter Benutzer sein eigenes System als normalen Prozeß hochfahren und austesten. Unter anderem können so kritische Hurd-Server auf einem anderen Terminal »von außen« mit dem Debugger »gdb« überwacht werden. In seinem eingekapselten Hurd-System erscheint es dem Benutzer, als wäre er der Administrator eines eigenen Rechners. Nach außen dringen jedoch keine speziellen Privilegien durch. Auch die Benutzung der Hardware ist natürlich eingeschränkt.

Hurd ist benutzbar

Das Hurd-System ist in seiner momentanen Entwicklungsversion durchaus benutzbar. Als Beweis dienen zum Beispiel die mehreren Hundert für das Hurd-System verfügbaren Debian-Pakete. Es ist auch in weiten Teilen stabil, aber es sind genug Fehler vorhanden, damit es nicht langweilig wird. Jeder ist herzlich willkommen, Hurd ausgiebig zu testen und uns von seinen Erfahrungen zu berichten. Wer mitmachen möchte, wird viele spannende Bereiche finden, in denen das Hurd-System erweitert werden kann. Als Beispiele seien nur PPP, IPv6, Bindungen zu diversen Sprachen, Portierung großer Softwarepakete, Portierung zu anderen Rechnerarchitekturen als ix86 und Entwicklung neuer Übersetzer für populäre Dateisysteme genannt, es gibt aber vieles mehr.

Wer jetzt auf den Geschmack gekommen ist und Hurd ausprobieren möchte, dem empfehle ich die von mir kompilierten und getesteten Binary-Pakete des sich momentan in der Entwicklung befindenden Debian GNU/Hurd-Systems. Ausführliche Installationsanleitungen finden sie auf der Debian GNU/Hurd-Homepage. Bei Problemen können Sie sich an die Liste (englisch) oder direkt an mich wenden (deutsch, englisch oder scheme).

```
This is the Hurd. Welcome.
```

■ *(uh)*

Der Autor:

Marcus Brinkmann studiert Mathematik an der Ruhr-Universität Bochum und arbeitet seit 1996 mit und an freier Software.