Proseminar „Speicher-und Dateisysteme“

Betriebssystemschichten

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Name: Dominik Downarowicz

Studiengang: Wirtschaftsinformatik

Betreuer: Michael Kuhn

Elmshorn, den 21.09.2015

Gliederung:

1 Einleitung

In Folge dieser schriftlichen Ausarbeitung widmet man sich dem Thema „Betriebssystemschichten“. Dabei wird einführend über das Betriebssystem an sich gesprochen, wie man es definiert und welche Aufgaben und Funktionen ein Betriebssystem hat. Danach wird der Kernel, der wichtigste Bestandteil eines Betriebssystems, vorgestellt und es wird gezeigt, wie man den Kernel klassifizieren kann, also welche Arten es gibt und wie sie sich untereinander unterscheiden. Passend dazu wählt man Beispiele zu den Kernelarten und stellt sie kurz vor. Anschließend wird noch die Verbindung zum Hauptthema „Speicher-und Dateisysteme“ gezogen, namentlich die Dateisysteme, indem man kurz über die Dateisysteme im Allgemeinem spricht und einige wichtige Beispiele benennt. Abschließend kommt noch eine Zusammenfassung aller behandelter Unterthemen.

2 Einführung Betriebssysteme

Wenn man über den Begriff „Betriebssystem“ nachdenkt, fallen den Menschen, die sich als geneigter PC-Nutzer sehen, verschiedene Bedeutungen und Funktionen ein. Dabei zeigt sich, dass es den meisten von ihnen schwer fällt, eine (relativ) eindeutige Bedeutung, eine (relativ) eindeutige Definition zu benennen. Was ist das Betriebssystem, wenn man es definieren müsste? Nach Din44300 gilt: „Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und die insbesondere die Abwicklung von Programmen steuern und überwachen“.

Die Definition umfasst unter anderem folgende Funktionen: Das Betriebssystem ist dafür zuständig, die Systemressourcen zu verwalten. Zu den Systemressourcen zählen der Arbeitsspeicher, die Festplatte und Ein-und Ausgabegeräte. Um es genauer zu bezeichnen, kann man sagen, dass das Betriebssystem die Systemressourcen an die verschiedenen Anwendungsprogramme verteilt und sie ihnen zur Verfügung stellt. Was daraus resultiert, ist die Tatsache, dass das Betriebssystem die Schnittstelle zwischen der Hardware und den Anwendungsprogrammen bildet.
Das klingt zwar alles wohl strukturiert, es lässt sich aber sagen, dass es letztendlich kein allgemeines Schema für den Aufbau eines Betriebssystems gibt. Ein Grund dafür sind die verschiedenen Systemfamilien und die verschiedenen Kernelarten.

3 Aufgaben eines Betriebssystems

Das Betriebssystem hat verschiedene wichtige Aufgaben. Unter anderem ist das Betriebssystem dafür bekannt, dass es die Fähigkeit hat, mehrere Aufgaben gleichzeitig auszuführen. Diese Fähigkeit ist allgemein bekannt unter dem Namen „Multitasking“. Dabei geht es nicht nur darum, dass diese Aufgaben gleichzeitig ausgeführt werden können, mehrere Anwendungen gleichzeitig offen gehalten werden können und man zwischen ihnen hin und her wechseln kann, sondern auch um die Fähigkeit, Aufgaben im Hintergrund zu bearbeiten, während eine andere Anwendung offen ist, was man als eine weitere wichtige Eigenschaft des „Multitasking“ bezeichnen kann.
Eine weitere wichtige Aufgabe des Betriebssystems findet sich in der Speicherverwaltung. Dabei geht es um die Verwaltung des Arbeitsspeichers und die Verteilung des Arbeitsspeichers an die einzelnen Prozesse. Eine Aufgabe, die damit in Verbindung steht, ist die sogenannte Segmentierung. Der virtuelle Speicher wird in einzelne Segmente unterteilt (daher der Name). Dadurch kann mehr Speicher adressiert werden, als im RAM tatsächlich zur Verfügung steht. Die Segmente werden auf die Festplatte ausgelagert und wieder in den Arbeitsspeicher geholt, wenn eine Anwendung sie benötigt.
Eine der wichtigsten Aufgaben ist die Verwaltung von Dateien. Das passiert in der Regel durch Dateisysteme. Ein Dateisystem ist ein Model, mit dem die Daten verwaltet werden. Dabei werden zwei Stufen benutzt: das konkrete Dateisystem und das virtuelle Dateisystem, dass die konkreten Dateisysteme verwaltet. Diese Stufen werden benutzt, da die meisten modernen Betriebssysteme mehrere konkrete Dateisysteme verwenden und der Verwaltungsaufwand dadurch minimiert wird.

4 Der Kernel

Wie man sieht, hat das Betriebssystem verschiedene Aufgaben. All das würde aber ohne den wichtigsten Bestandteil des Betriebssystems nicht funktionieren: Der Kernel. Der Kernel ist, wie der Name schon impliziert, der Betriebssystemkern und der zentrale Bestandteil des Betriebssystems. Im engeren Sinne wird der Kernel als das eigentliche Betriebssystem gesehen, wenn es nach Andrew S. Tanenbaum geht:
„Editoren, Compiler, Assembler, Binder und Kommandointerpreter sind definitiv nicht Teil des Betriebssystems, auch wenn sie bedeutsam und nützlich sind“.
Die oben genannten Aufgaben eines Betriebssystems spiegeln sich mehr oder weniger im Aufbau eines Kernels wider. Der Kernel ist in Schichten aufgebaut und diese Schichten umfassen: Die Schnittstelle zur Hardware, die Speicherverwaltung, die Prozessverwaltung, die Geräteverwaltung und die Dateisysteme.
Das ist der allgemeine Aufbau eines Kernel. In der Realität gibt es aber verschiedene Kernelarten. Bevor die Kernelarten vorgestellt werden, wird noch ein weiteres wichtiges Konzept bezüglich des Kernels erläutert.

4.1 Kernel-und Benutzermodus

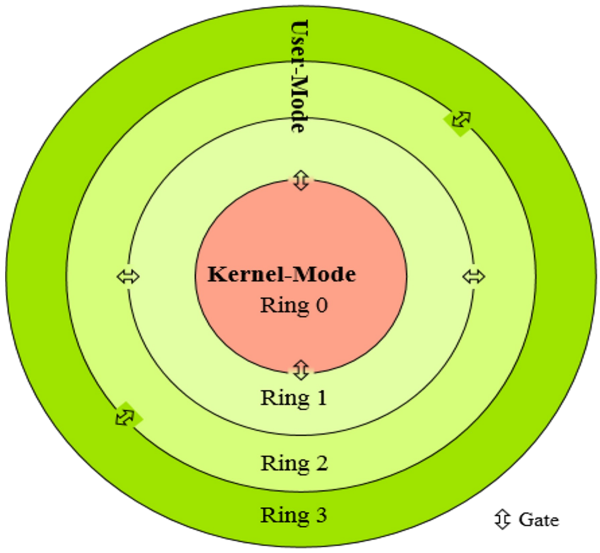

Abbildung 4.1 Der Ringaufbau

Bildquelle: http://de.wikipedia.org/wiki/Ring_%28CPU%29

Der Ring zeigt eine Sicherheits-und Privilegierungsstufe einer Aufgabe, die innerhalb eines Prozesses zusammengefasst wird. Insgesamt gibt es vier Privilegierungsstufen. Im Zentrum des Rings (Kernelmodus) läuft das Betriebssystem und die äußeren Ringe (Benutzermodus) sind für die Anwendungen gedacht. Die Prozesse bleiben im jeweiligen Ring und wenn eine Kommunikation notwendig ist, findet sie über die Gates statt. Und die Kommunikation ist zweifelsohne notwendig, da nur Ring 0, also der Kernelmodus, den vollen Zugriff gestattet.

Nun fragt man sich, wieso es diese Unterscheidung zwischen den Stufen, die Unterscheidung zwischen dem Kernel-und Benutzermodus gibt.
Die Gründe dafür sind unter anderem folgende:
Durch die Unterscheidung werden Prozesse voneinander abgeschottet.
Inwiefern bringt das einen Nutzen? Nun, zum einen werden dadurch fehlerhafte Prozesse abgeschottet, die deshalb keine weiteren Prozesse stören oder sogar das ganze System zum erliegen bringen. Zum anderen wird der Zugang für unterprivilegierte Prozesse sowohl beschränkt als auch ermöglicht. Unterprivilegierte Prozesse können den Zugang bekommen, wenn sie ihn benötigen oder ihnen wird der Zugang verwehrt, wenn sie ihn nicht erhalten dürfen. Ein weiterer Grund ist die Trennung der Speicherbereiche. Dadurch wird Sicherheit und Stabilität gewährleistet.

Warum bekommt der Kernel-und Benutzermodus diese Vorstellung und Erwähnung? Das liegt daran, dass die verschiedenen Kernelarten auf die unterschiedliche Benutzung dieses Konzepts basieren.

4.2 Die Kernelarten

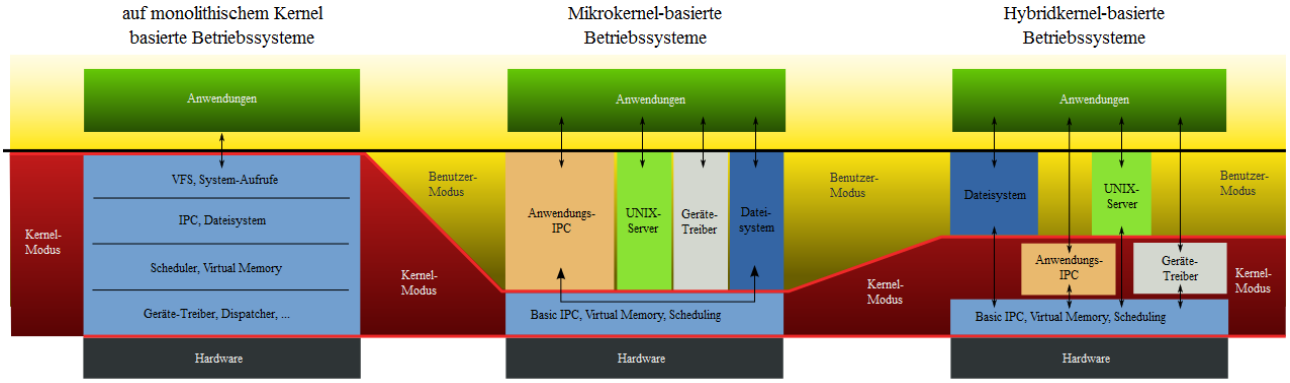

Abbildung 4.2 Die Kernelarten gegenübergestellt

Bild geändert: http://de.wikipedia.org/wiki/Monolithischer_Kernel

Der Kernel lässt sich in drei verschiedene Arten unterteilen: Der Monolithische Kernel, der Mikrokernel und der Hybridkernel. Sie alle verwenden das Konzept des Kernel-und Benutzermodus anders.

Da wäre zum einen der Monolithische Kernel. Bei dieser Art befinden sich jegliche Betriebssystemkomponente im Kernelmodus. Dazu gehören die Speicher-und Prozessverwaltungsfunktionen, die Kommunikation zwischen den Prozessen, die Gerätetreiber und weitere Funktionen. Die Anwendungen laufen wie gehabt im Benutzermodus. Diese Kernelart kann man wohl als den klassischen Kernel (zumindest in der Vergangenheit) betrachten. Ein Beispiel dafür wäre Linux.

Zum anderen gibt es den Mikrokernel. In diesem Fall laufen nur wenige Betriebssystemkomponente im Kernelmodus wie z.B. Grundfunktionen und die Speicher-und Prozessverwaltung. Im Benutzermodus laufen hingegen die Anwendungs-IPC, die Gerätetreiber und die Dateisysteme. Diese Kernelart ist das Gegenteil zum Monolithischen Kernel und ein Beispiel dafür wäre Minix.

Zu guter Letzt gibt es den Hybridkernel. Wie der Name schon sagt, ist diese Kernelart eine Verbindung der beiden anderen Kernelarten. Ein Kompromiss, der versucht, die Vorteile des Monolithischen Kernels und des Mikrokernels zu vereinen. In diesem Fall gibt es keine allgemeine Struktur, wie ein Hybridkernel auszusehen hat, da man unterschiedliche Wege gehen kann, um das Optimum der Vorteile zu erreichen. Beispiele dafür sind das Mac OS X und Windows.

Die Vor-und Nachteile der Kernelarten sind vielfältig. Der Monolithische Kernel hat unter anderem zum Vorteil, dass keine aufwendige Kommunikation notwendig ist, da alle wichtigen Prozesse ohnehin im Kernelmodus stattfinden.

Außerdem weist der Monolithische Kernel eine gute Performance auf, was darauf zurückzuführen ist, dass nur ein minimaler Aufwand beim Wechsel zwischen den Privilegierungsstufen entsteht. Zum Nachteil hat der Monolithische Kernel, dass das Auswechseln von Komponenten ziemlich aufwendig wird. Darauf aufbauend gilt das gleiche für Funktionsänderungen, da man bei solchen Änderungen den Kernel neu übersetzen muss. Zudem kann es vorkommen, dass fehlerhafte Systemkomponente zum Komplettausfall führen, da sich alle Betriebssystemkomponente im Kernelmodus befinden.
Der Mikrokernel hat zum Vorteil, dass dessen Komponente separiert sind und deshalb ein Austausch ohne Beeinträchtigung stattfinden kann. Außerdem besteht ein hoher Ausfallschutz, da nur wenige Komponente im Kernelmodus sind. Zudem befinden sich die Treiber im Benutzermodus. Warum ist das ein Vorteil? Nun, Da die Treiber im Benutzermodus laufen, können die Zugriffsrechte einzeln bestimmt werden. Die Treiber müssen nicht im Kernelmodus sein und können mit Benutzerrechten ausgeführt werden, was erheblich einfacher ist. Die Nachteile sind unter anderem ein Geschwindigkeitsverlust durch den Wechsel zwischen den Modi und eine schwierige Synchronisation der Nutzerprozesse.
Wenn es um den Hybridkernel geht, kann man nicht eindeutig sagen, welche Vor-und Nachteile sich mit Sicherheit durchsetzen. Immerhin ist der Hybridkernel der Kompromiss zwischen den anderen Kernelarten.

5 Beispiel Linux

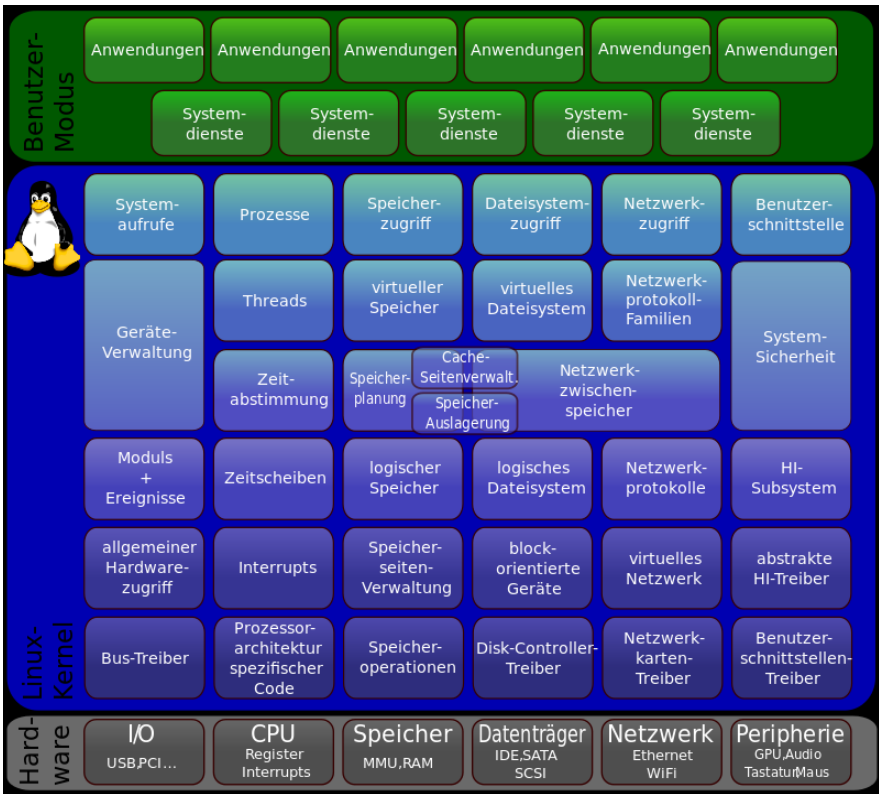

Abbildung 5.1 Der Linux Kernel

Bildquelle: http://de.wikipedia.org/wiki/Linux_%28Kernel%29

Der Linux Kernel; das wahrscheinlich bekannteste Beispiel für einen monolithischen Kernel. Wie andere Kernel, hat auch der Linux-Kernel die Funktion, die Schicht zu bilden, die sich zwischen der Hardware und der Software befindet, was man an der Abbildung 5.1 erkennen kann. Er bildet die einheitliche Schnittstelle für die Software, was zum Vorteil hat, dass die Software, die die Hardware nutzen will, die Hardware nicht genauestens kennen muss. Als monolithischer Kernel übernimmt Linux die folgenden Funktionen: Speicherverwaltung, Prozessverwaltung, Multitasking, Lastverteilung, und Eingabe/Ausgabe-Operationen auf verschiedenen Geräten.
Wenn man von Linux als monolithischen Kernel spricht, muss man einen wichtigen Unterschied feststellen. Linux ist nämlich kein strikt monolithischer Kernel. Ein strikt monolithischer Kernel zeigt sich dadurch, dass der gesamte Quellcode und alle Treiber im Kernel kompiliert sind. Linux hat aber eine zusätzliche Besonderheit. Linux kann sogenannte Module bei Bedarf laden oder wieder entfernen. Diese Module und ihre dynamische Natur erzeugen eine gewisse Flexibilität, die einem strikt monolithischen Kernel fehlt. Es müssen nicht alle Treiber im Speicher gehalten werden, da sie als Module je nach Erfordernis eingesetzt werden können, um verschiedenste Hardware anzusprechen. Diese Tatsache gilt für nahezu alle Treiber, ausgenommen Treiber, die für die Inbetriebnahme des Systems nötig sind.
Wenn man sich zurückerinnert, wurde bei dieser Ausarbeitung das Konzept vom Kernel-und Benutzermodus vorgestellt. Im Falle von Linux laufen die Treiber, die sich im Kernel befinden, und Kernelmodule im Kernelmodus, also folglich im Ring 0. Sie haben vollen Zugriff. Ansonsten befinden sich hier und da ein paar Module im Benutzermodus, im Ring 3. Die Ringe 1 und 2 werden von Linux nicht benutzt, was nicht so unüblich ist, dass nicht alle Ringe der Ringarchitektur benutzt werden. Eine weitere Besonderheit ist, dass Linux mittlerweile Unterstützung von FUSE und anderen Erweiterungen bekommt, sodass man sagen kann, dass Ideen des Mikrokernels in den monolithischen Kernel von Linux einfließen.

6 Beispiel Minix

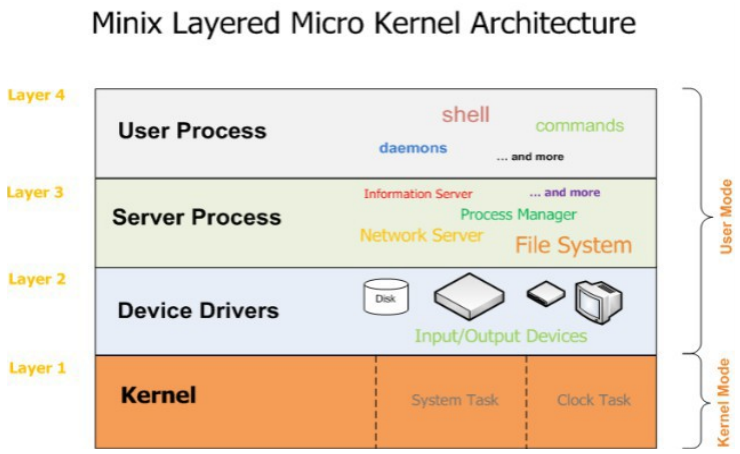

Abbildung 6.1 Die Minix Schichten Architektur
Bildquelle: https://imma.wordpress.com/2007/04/02/presentation-internal-structure-of-minix/

Als Gegensatz zu Linux, ist Minix ein Mikrokernel. Entwickelt wurde Minix von Andrew S. Tanenbaum als Gegenbeispiel zu Linux. Wie man an der Abbildung 6.1 erkennen kann, ist Minix in Schichten aufgebaut. Die unterste Schicht bildet der Kernel, der für die Grundfunktionen und dafür, dass das System in erster Linie überhaupt läuft, zuständig ist. Nur diese Schicht befindet sich im Kernelmodus. Die anderen Schichten, die die Gerätetreiber, die Dateisysteme und diverse Prozesse beinhalten, sind im Benutzermodus. Wie schon genannt, hat diese Architektur Vor- und Nachteile. Einerseits ist eine hohe Sicherheit gewährleistet, da sich nur eine Schicht im Kernelmodus befindet und somit die Chance eines Komplettausfalls minimiert wird.

Zudem können Erweiterungen bei Bedarf einfach hinzugezogen oder entfernt werden, ohne große Änderungen vorzunehmen. Und durch die klare Struktur wird eine gewisse Stabilität erzeugt. Auf der anderen Seite ist natürlich die Kommunikation zwischen den Prozessen aufwändig und die Realisierung dieses Konzeptes ist anfangs schwierig umzusetzen. Jedoch kann man sagen, dass Minix ein gutes Beispiel für einen Mikrokernel ist. Für den entsprechenden Anwendungsbereich funktioniert diese Architektur. Zudem gilt, dass Minix sowohl eine Inspiration, als auch ein Gegenbeispiel für Linux ist.

7 Beispiel Mac OS X

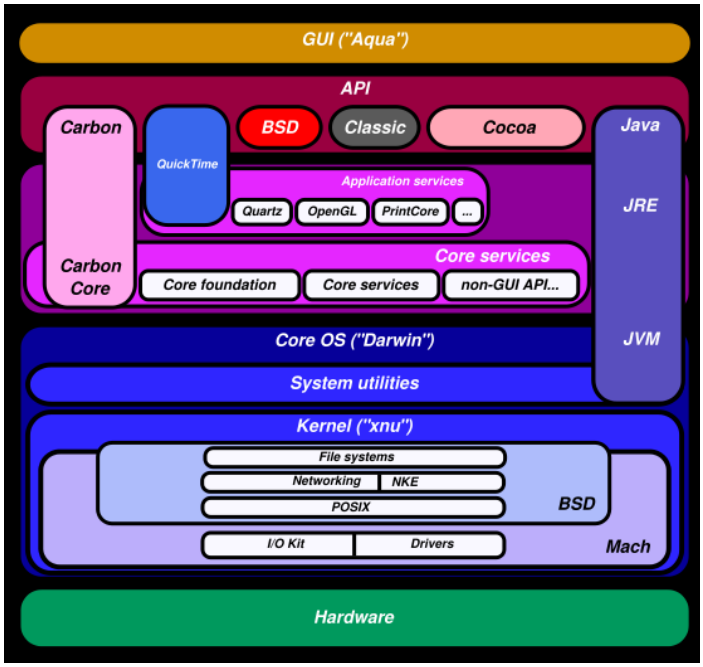

Abbildung 7.1 Das Mac OS X

Bildquelle:
http://de.wikibooks.org/wiki/MacOSKompendium/_Unter_der_Haube_von_Mac_OS_X:_UNIX/_Architektur_von_Mac_OS_X

Zu guter Letzt wird hier Mac OS X kurz vorgestellt. Das von Apple entwickelte Betriebssystem ist in abgewandelter Form auch als iOS bekannt. Mac OS X ist ein Beispiel für einen Hybridkernel. Warum? Nun, es setzt auf einen Hybridkernel.

Wie man an der Abbildung 7.1 erkennen kann, ist Mac OS X in fünf Schichten aufgebaut. Von unten nach oben: Die Hardware, das Kernsystem Darwin, die Kerndienste, die Medienschicht und die Anwendungsschicht. Besondere Erwähnung erhält der XNU Kernel. Dieser Kernel sollte den reinen Mach-Mikrokernel ersetzen, indem der bestehende Mikrokernel durch zusätzliche Teile erweitert wird, jedoch nicht so viele Funktionen wie ein monolithischer Kernel haben soll.

Die zusätzlichen Teile wurden durch den monolithischen BSD- Kernel ergänzt, wodurch ein Kompromiss entstanden ist: Ein Hybridkernel. Als Basis gilt der Mach-Mikrokernel, der fürs Multitasking, die Speicherverwaltung und die Fehlerbehandlung zuständig ist und grundlegende Gerätetreiber enthält. Der BSD Kernel hingegen ist für die Benutzerverwaltung, die Dateirechte und Standarddateisysteme zuständig. Dieser Hybridkernel nutzt die zwei oben genannten Kernel und versucht, Vorteile beider Kernelarten zu vereinen.

8 Dateisysteme

Nachdem jetzt die verschiedenen Kernelarten vorgestellt wurden, gehe ich auf die Dateisysteme ein und wie ihre Relation zum Betriebssystem ist. Wie schon gesagt ist die Verwaltung der Dateisysteme Aufgabe der Betriebssysteme. Mehr noch, die Dateisysteme sind Bestandteil des Betriebssystems. Aber was machen die Dateisysteme eigentlich? Kurz gefasst, sind sie dafür verantwortlich, Daten zu benennen, zu speichern und zu organisieren. Es gibt eine Menge verschiedener Daten, verschiedene Standards und verschiedene Anforderungen. Aus diesem Grund sind viele verschiedene Arten von Dateisystemen entstanden. Hier werden die wichtigsten von ihnen in Bezug auf das Betriebssystem vorgestellt.

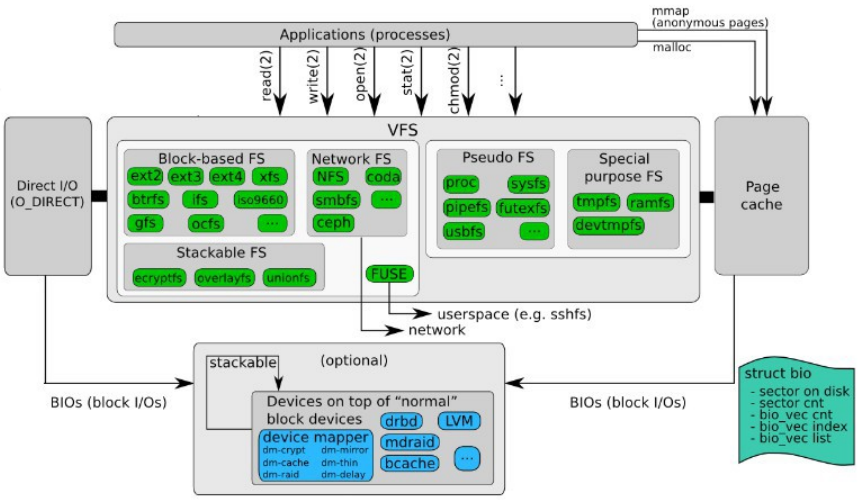

Abbildung 8.1 Virtual file system switch
Bild geändert: https://www.thomas-krenn.com/de/wiki/Linux_Storage_Stack_Diagramm

Eins der wichtigsten Beispiele eines Dateisystems ist das Virtuelle Dateisystem oder auch das VFS switch (Virtual File System switch). Das Virtuelle Dateisystem ist eine Kernel Software Schicht, die dafür zuständig ist, eine einheitliche Schnittstelle für Programme zur Verfügung zu stellen, um auf die diversen konkreten Dateisysteme zuzugreifen. Die Abbildung 8.1 zeigt diese Eigenschaft einer Schnittstelle. Demnach sind beide Arten von Dateisystemen, die eigentlichen Dateisysteme und das Virtuelle Dateisystem, das für die Verwaltung und den Zugriff zuständig ist, essentiell. Das Virtuelle Dateisystem verdeckt die konkreten Dateisysteme, was zum Vorteil hat, dass die Programme nicht genau wissen müssen, auf welches Dateisystem sie zugreifen müssen. Und dass ist ein nicht zu unterschätzender Vorteil, da bis zu 40 Dateisysteme unterstützt werden wie z.B. FAT, NFS oder /proc-fs.

Wenn es um die Bestandteile eines Virtuellen Dateisystems geht, können vier wichtige Objekte unterschieden werden. Da wäre zum einen der Superblock. Das ist das grundlegendste Objekt des Virtuellen Dateisystems, da durch die Zerstörung oder Beschädigung dieses Objekts das gesamte Dateisystem unlesbar wird. Dann wäre da das Inode-Objekt. Je Datei gibt es ein Inode und die Inode-Nummer. Mit der Inode-Nummer kann man die Datei eindeutig identifizieren. Zudem enthält das Inode-Objekt allgemeine Informationen wie z.B. den Besitzer, die Zugriffsrechte, die Operationen und Dateisystem spezifische Informationen.Ein weiteres wichtiges Objekt ist das File-Objekt. Es existiert zur Laufzeit im Kernelspeicher und nur solange die Datei offen ist. Dieses Objekt enthält Informationen zur Interaktion zwischen einer geöffneten Datei und einem Prozess. Und zuletzt gibt es noch das Dentry-Objekt. Dieses Objekt entsteht, wenn ein Verzeichnis geladen wird. Das Virtuelle Dateisystem verwandelt das Verzeichnis in das Dentry-Objekt. Dieses speichert die Verbindung zwischen dem Verzeichniseintrag und der Datei und wird dafür benutzt, um neue Dateisysteme einzubinden. Das sind alles in allem die wichtigsten Objekte in Bezug auf das Virtuelle Dateisystem.

Eine Anmerkung sollte noch gemacht werden. Das Virtuelle Dateisystem bzw. das VFS switch sollte nicht mit den speziellen virtuellen Dateisystemen verwechselt werden. Diese zählen zu den eigentlichen Dateisystemen, die unter dem VFS switch als einheitliche Schnittstelle vereinigt werden. Sie enthalten nur rein virtuell vorkommende Dateien mit Informationen, die auf eine Datei abgebildet werden. In Wirklichkeit werden diese Dateien vom Kernel zur Verfügung gestellt und befinden sich auf keiner Festplatte. Ihre Existenz wird vom Kernel „vorgegaukelt“.

Ein weiteres wichtiges Beispiel für Dateisysteme ist FUSE (Filesystem in Userspace). Dieses Dateisystem ermöglicht es, Dateisysteme in den Benutzermodus zu verlagern. Dadurch können die Beschränkungen vom Kernelmodus umgangen werden, da man schon mit Benutzerrechten eigene Dateisysteme einbinden kann.

Um nochmal den Unterschied zwischen dem Virtuellem und den speziellen virtuellen Dateisystemen zu zeigen, werden hier kurz zwei Dateisysteme dieser Art vorgestellt.
Eins dieser Dateisysteme ist /proc-fs. Passend zur Grundeigenschaft von den speziellen virtuellen Dateisystemen, braucht /proc-fs keinen direkten Speicher. Es wird dynamisch vom Kernel erzeugt und hat Informationen zu Prozessen wie z.B. das Arbeitsverzeichnis oder den virtuellen Speicher des Prozesses. Zudem kann dieses Dateisystem Informationen zur CPU haben.
Ein weiteres Dateisystem, dass man zu den speziellen virtuellen Dateisystemen zählen kann, ist tmpfs. Zwar existieren die Dateien in diesem Fall, jedoch ist alles Gespeicherte nur temporär, da dieses Dateisystem ausschließlich das RAM nutzt. Das hat zum Vorteil, dass die Geschwindigkeit erhöht wird, jedoch es zum Totalverlust kommen kann, wenn ein Systemabsturz stattfindet, was aus der Eigenschaft resultiert, dass die Dateien temporär sind.
Das sind nur ein paar Beispiele für Dateisysteme.

9 Zusammenfassung

Zusammenfassend lässt sich sagen, dass mehr hinter dem Thema „Betriebssystemschichten“ steckt, als man denkt. Und diese Ausarbeitung hat nur einen kleinen Teil gezeigt, da sich sehr viel mehr darüber erzählen lässt. Diese Ausarbeitung zeigt, dass das Betriebssystem viele wichtige Aufgaben hat und der wichtigste Bestandteil eines Betriebssystems der Kernel ist. Den Kernel kann man in drei verschiedene Arten unterteilen. Den Monolithischen Kernel, den Mikrokernel und den Hybridkernel. Jede dieser Kernelarten hat Vor-und Nachteile. Es lässt sich aber mit Sicherheit sagen, dass der Fokus auf den Hybridkernel liegen wird, da genau diese Kernelart die Vorteile beider anderer Kernelarten vereint und einen möglichst optimalen Kompromiss erzeugt. Beispiele dieser Kernelarten lassen sich überall finden. Der Linux-Kernel ist ein Beispiel für einen monolithischen Kernel, der aber noch die zusätzliche Eigenschaft der Module, die bei Bedarf geladen und wieder entfernt werden können, hat. Minix ist ein Beispiel für einen Mikrokernel. Er besteht aus fünf Schichten und nur die unterste Schicht ist im Kernelmodus, was einen höheren Ausfallschutz und bessere Möglichkeiten für das Ein-und Ausbauen von Komponenten erzeugt. Und Mac OS X, das einen Hybridkernel benutzt, der aus einem Mikrokernel und Teilen eines Monolithischen Kernels aufgebaut ist. Das sind nur drei Beispiele. Zum Abschluss wurden Dateisysteme erläutert und dass das Virtuelle Dateisystem oder VFS switch eine wichtige Schicht des Kernels ist, die als Schnittstelle für die konkreten Dateisysteme gilt und nicht mit den speziellen virtuellen Dateisystemen verwechselt werden soll, die zumeist ebenfalls innerhalb des VFS switch zusammengefasst werden.

Hoffentlich konnte diese Ausarbeitung einen ungefähren Einblick in diese Thematik und ihre Konzepte gewähren.

10 Quellen

Baun, Dr. Christian: „7. Vorlesung Betriebsysteme“: Benutzermodus und Kernelmodus. Stand: 2012/2013. http://baun-vorlesungen.appspot.com/BTM1213/Skript/folien_bts_vorlesung_07_WS1213.pdf (9. September 2015).

„Betriebssystem“. Stand: 16. September 2015. http://de.wikipedia.org/wiki/Betriebssystem (9. September 2015).

„Betriebssystemtheorie/ Einleitung“. Stand: 15. Februar 2013. http://de.wikibooks.org/wiki/Betriebssystemtheorie/_Einleitung (9. September 2015).

Duwe, Kira Isabel: „Comparison of kernel and user space file systems“: 2. Kernel file systems. Stand: 28. August 2014. http://edoc.sub.uni-hamburg.de/informatik/volltexte/2015/210/pdf/bac_duwe.pdf (9. September 2015).

Fischer, Werner / Schönberger, Georg: „Linux Storage Stack Diagramm“. Stand: 8. Juli 2015. https://www.thomas-krenn.com/de/wiki/Linux_Storage_Stack_Diagramm (9. September 2015).

„Hybridkernel“. http://de.wikipedia.org/wiki/Hybridkernel (9. September 2015).

http://www.inf.fu-berlin.de/lehre/SS01/OS/LabTalks/filesystem.pdf (9. September 2015).

„Kernel (Betriebsystem)“. Stand: 21. September 2015. http://de.wikipedia.org/wiki/Kernel_%28Betriebssystem%29 (9. September 2015).

Kersken, Sascha: „Kompendium der Informationstechnik“: Kapitel 4 Betriebsysteme: 4.2 Aufgaben und Konzepte. http://openbook.rheinwerk-verlag.de/kit/itkomp04001.htm#Rxx355kap04001040002021F04F19F (9. Seprember 2015).

„Linux (Kernel)“. Stand: 21. September 2015. http://de.wikipedia.org/wiki/Linux_%28Kernel%29 (9. September 2015).

„Mac-OS-Kompendium/ Unter der Haube von Mac OS X: UNIX/ Architektur von Mac OS X“. Stand: 22. Juli 2013. http://de.wikibooks.org/wiki/Mac-OS-Kompendium/_Unter_der_Haube_von_Mac_OS_X:_UNIX/_Architektur_von_Mac_OS_X (9. September 2015).

„Microkernel“. Stand: 1. Juli 2014. http://de.wikipedia.org/wiki/Microkernel (9. September 2015).

„Monolithischer Kernel“. Stand: 4. September 2015. http://de.wikipedia.org/wiki/Monolithischer_Kernel (9. September 2015).

„Presentation: Internal Structure of Minix“. Stand: 2. April 2007. https://imma.wordpress.com/**2007/04/02/presentation-internal-structure-of-minix/.** (9. September 2015).

„Ring (CPU)“. Stand: 16. September 2015. http://de.wikipedia.org/wiki/Ring_%28CPU%29 (9. September 2015).