Proseminar

„Speicher- und Dateisysteme“

Betriebssystemschichten

11.03.2011

Bernd Ihnen

Gliederung

1 Einleitung

Jeder arbeitet mit ihnen und jeder hat eine Meinung über sie – Betriebssysteme! Im Laufe der Zeit haben sich verschiedene Technologien und Betriebssystem(kern)konzepte entwickelt. Doch was verbirgt sich hinter den Betriebssystemen und wie werden sie realisiert? Nach einer kurzen Themeneinführung gibt diese Arbeit Auskunft über den grundsätzlichen strukturellen Aufbau ausgewählter Betriebssystemkerne bzw. –kernel sowie deren Vor- und Nachteile. In Kapitel 4 wird die nähere Umgebung der Dateisysteme in einem Linuxsystem beschrieben sowie die Komponenten des Betriebssystems MINIX genannt, dessen Betriebssystemkern vom schematischen Aufbau in Kontrast zu einem Linuxkern steht. Die abschließende Zusammenfassung zieht ein kurzes Résumé und weist auf markante Unterschiede der Konzepte hin.

2 Themeneinführung

Die Geschichte der Computertechnologie begann Ende des Zweiten Weltkrieges (bereits vorher hatte es konzeptionelle Ideen gegeben). Heute leben wir in der Vierten Generation der Personalcomputer. Die Hardware eines modernen Computers besteht u.a. aus Prozessoren, Arbeitsspeicher, Festplatten, Bildschirm, Tastatur, Maus, usw., die über Systembusse verbunden sind. Damit diese Geräte einfach genutzt werden können, ohne dass jeder Anwendungsprogrammentwickler diese Hardware direkt anspricht und damit die recht komplizierte Maschinensprache beherrschen muss, dienen Betriebssysteme im Wesentlichen als erweiterte Maschine und als Ressourcenverwalter der Hardware.

Gemäß **DIN 44300** wird ein Betriebssystem wie folgt definiert:

*„**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**."*

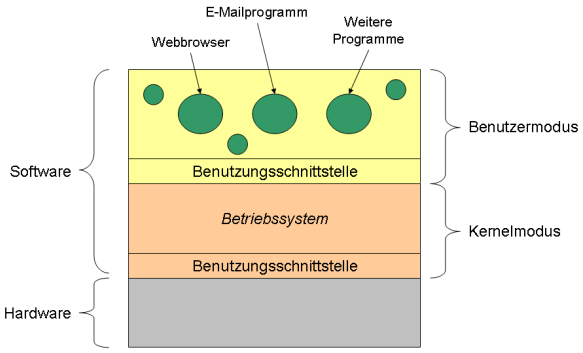

Abbildung 2.1

Abbildung 2.1 zeigt einen stark abstrahierten Aufbau eines Computers. Die Oberfläche, die der Benutzer verwendet, kann eine textbasierte (Shell) oder grafische Benutzungsschnittstelle (Graphical User Interface bzw. GUI) sein und ist nicht Teil des Betriebssystems! Darauf aufbauend werden gängige Software wie z.B. Office-, Multimedia Produkte oder Webbrowser installiert.

Konvention: Bei den folgenden Abbildungen werden die Hardware-Elemente grau/ schwarz, die Kernelschichten orange und Benutzerschichten gelb bis grün dargestellt.

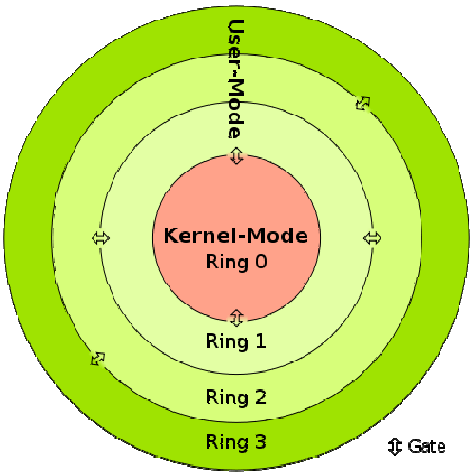

Abbildung 2.2

Abbildung 2.2 zeigt den **Ringaufbau** eines x86-Prozessors. Innerhalb dieser Schichten gibt es verschiedene Zugriffsrechte auf die Hardwareprozesse. Dies ist sinnvoll, um von der Hardware zu abstrahieren und um Prozessen voneinander zu trennen. Gates (Schnittstellen) sind eine der Möglichkeiten zur Kommunikation auch über die Ringgrenzen hinaus. Im **Ring 0** befindet sich der Kernel-Modus, der direkt auf die Hardware zugreifen kann. Dies ist der **Betriebssystemkern** oder –kernel. Darauf aufbauende Ringe 1 bis 3 erhalten zunehmend weniger Rechte auf die Speicherblöcke der unteren Ringe und somit auch auf die Komponenten des Betriebsystemkerns. Diese werden daher User-Modus genannt.

Abstraktion und Modularisierung sind also bereits bei dem Aufbau von Betriebssystemen und der darüberliegenden Software sehr wichtige Konzeptionskomponenten. Sie dienen der Vereinfachung und Ressourceneinteilung.

3 Übersicht verschiedener Kernelarchitekturen

Ausgehend von dem hierarchischen Aufbau gemäß obiger Erläuterungen und Abbildungen 2.1 sowie 2.2, können verschiedene Kernelarchitekturen verwendet werden. Sie sind Bindeglied zwischen der Hardware und der Benutzungsoberfläche. Einen Überblick soll die Beschreibung bzw. der Vergleich zwischen den Kernelarchitekturen „Monolithischer Systeme“, „Mikrokerne“ und „Hybridkerne“ geben. Wie und ob die Betriebssystem-komponenten modularisiert sind, liegt an der Architektur des jeweiligen Betriebssystems.

Nachfolgend wird der Aufbau sowie Vor- und Nachteile eines Mikrokernels beschrieben. Anschließend folgt eine kurze Beschreibung eines monolithischen Systems unter Nennung der markanten Unterschiede zu einem Mikrokernel. Der Hybridkerne ist eine variable Architektur zwischen Mikrokernel und einem Monolithischen System. Welche Systemkomponenten im Benutzer- bzw. Kernel-Modus implementiert werden, liegt ebenfalls an der individuellen Architektur eines Betriebssystems. Natürlich gibt es neben diesen drei Kernelarchitekturen noch weitere. Diese würden jedoch nicht in den Rahmen dieser Arbeit passen. Einen kleinen Einblick liefert Andrew S. Tanenbaum in seinem Buch „Moderne Betriebssysteme“.

3.1 Mikrokernel

Der **Mikrokernel** ist eine Betriebssystemarchitektur, die möglichst wenige Betriebssystemkomponenten im Kernel-Modus hält. Ziel ist eine hohe Ausfallsicherheit und hohe Modularisierung seitens der Architektur. Abbildung 3.1 zeigt einen möglichen Aufbau eines Mikrokernels.

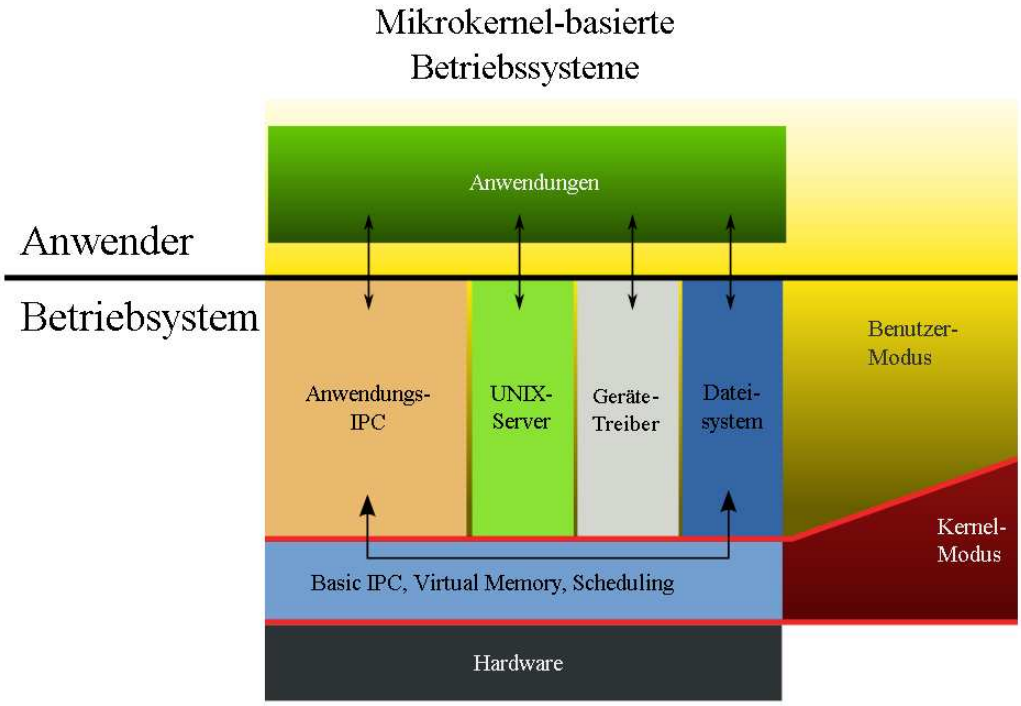

Abbildung 3.1

Im **Kernel-Modus,** direkt über der Hardware, befinden sich lediglich Basic-IPC (Basis-Interprozesskommunikation), Virtual Memory (virtueller Speicher) und das Scheduling (Prozess-Planung). Der Scheduler verwaltet die Prozesse eines Betriebssystems. Er bestimmt somit, welche Prozessanfrage in welcher Reihenfolge abgearbeitet wird. Dies kann präemtiv (unterbrechend) oder non-präemtiv (nicht unterbrechend) erfolgen. Durch präemtive Scheduler ist es möglich auf einem Einfach-Prozessor mehrere Prozesse scheinbar gleichzeitig abarbeiten zu lassen. Die IPC im engeren Sinne ermöglicht die Kommunikation zwischen Prozessen innerhalb eines Computers unter strikter Trennung des Speicherbereiches durch so genannte Pipes. Der Virtual Memory ist ein dem Betriebssystem zur Verfügung gestellter Adressraum, der vom tatsächlich vorhandenen Arbeitsspeicher unabhängig ist. Durch eine virtuelle Speicherverwaltung kann ein Betriebssystem einen Adressraum generieren, der für ein Anwendungsprogramm zusammenhängend dargestellt wird, physisch jedoch verteilt gespeichert ist. Der Vorgang der Abbildung der virtuellen Adressen auf die physischen wird Paging genannt.

Im **Benutzer-Modus** befinden sich Anwendungs-IPC (Anwendungs-Interprozesskommunikation), UNIX-Server, Geräte-Treiber und das Datei-System. Die

Anwendungs-IPC stellt im Benutzer-Modus dieselben Funktionen zur Verfügung wie die Basic-IPC im Kernel-Modus. Geräte-Treiber ermöglichen die Kommunikation zwischen Software und Hardware durch Nutzung entsprechender Schnittstellen, die Funktionen (z.B. Komprimierung, Verschlüsselung) bereitstellen. Eine der wichtigsten Aufgaben von Betriebssystemen ist die Darstellung eines oder mehrerer Dateisysteme. Je nach Betriebssystem können diese unterschiedliche Funktionen anbieten. Der Unix-Server stellt modular ein Subsystem einer Unix-Umgebung nach POSIX-Standard dar. Er erfüllt die Funktion einer standardisierten API (Application Programming Interface) und ermöglich den Anwendungsprogrammen somit eine Schnittstelle zum Betriebssystem.

Wie eingangs beschrieben, ist das **Ziel** eines Mikrokernels die Gewährleistung einer hohen Ausfallsicherheit. Diese wird durch die separaten Systemkomponenten mit klarem Schnittstellendesign erreicht. Ein Ausfall einer Komponente würde somit nicht zum Absturz des gesamten Systems führen. Zudem arbeiten die Gerätetreiber im Benutzermodus. In Anbetracht zahlreicher auf dem Markt existierender Hardwarekomponenten sowie Treiber, ist eine im Zusammenhang mit dem Betriebssystem (und dessen Stand) auftretende Fehlfunktion nicht immer vermeidbar. Z.B. würde ein Fehler des Audiotreibers somit nicht zum Absturz des ganzen Systems führen. Da die Komponenten gekapselt sind, soll als weiterer Vorteil der einfache Austausch einzelner Komponenten genannt sein.

Ein **Nachteil** ist der grundsätzliche Geschwindigkeitsverlust, der durch die Kommunikation über die Schnittstellen zwischen Kernel-Modus und Benutzer-Modus hinweg resultiert. Zudem ist ein hoher Synchronisationsaufwand der Benutzer-Prozesse aufzuführen. Die Koordination im Kernel-Modus wird dadurch komplex und schwer optimierbar.
Der Vorteil der Geräte-Treiber im Benutzer-Modus wirkt als Nachteil bei einigen Input-/ Outputzugriffen, die ohne bestimmte Rechte schwer zu implementieren sind. Der Hinweis zur Implementierung einiger Gerätetreiber bzw. –funktionen im Kernel-Modus würde nicht nur den theoretischen Aspekt des Abweichens von diesem Idealtyp anführen, sondern hätte auch negative Auswirkungen auf den o. g. Vorteil der Ausfallsicherheit, sofern dies nicht modularisiert erfolgt.

3.2 Vergleich Mikrokern - Monolithische Systeme

Monolithische Systeme zeichnen sich dadurch aus, dass sämtliche Betriebssystem-komponenten im Kernel-Modus arbeiten. Dies führt zu einem abweichenden Aufbau sowie angepasste Komponenten. Ziel dieser Architektur ist die Nutzung des Geschwindigkeitsvorteils. Abbildung 3.2 zeigt ein Schema eines Monolithischen Systems.

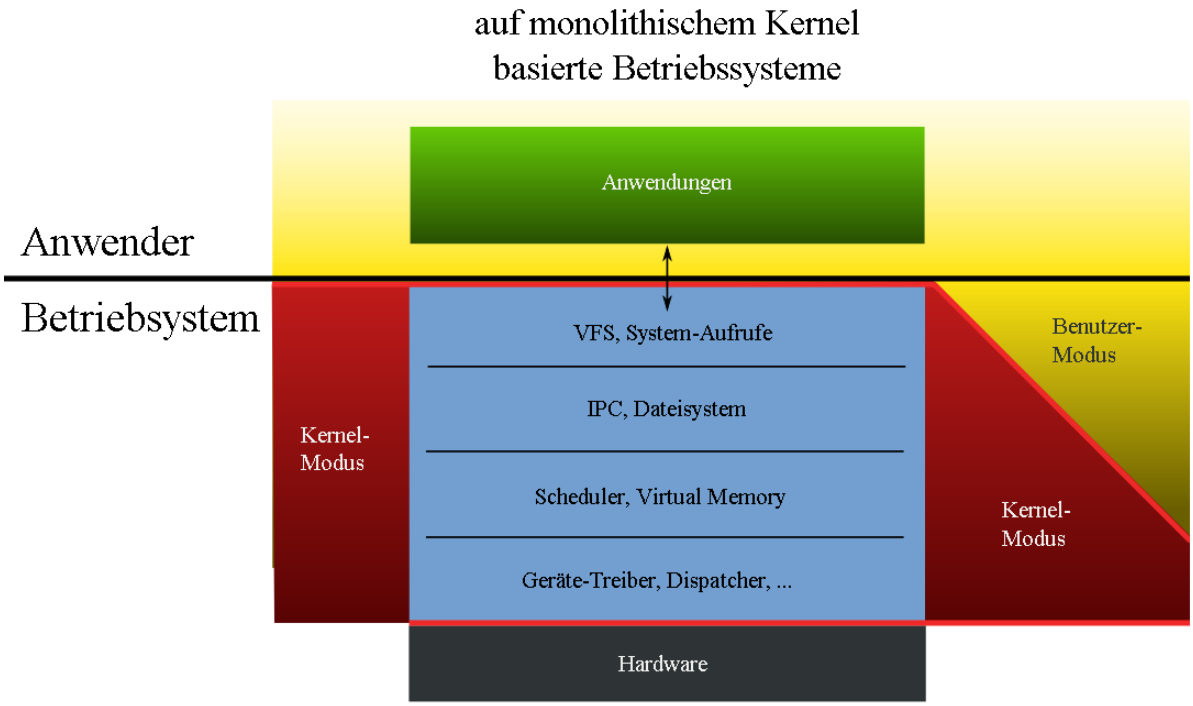

Abbildung 3.2

Bei einem Monolithischen System laufen alle Module im **Kernel-Modus**. Auf der Hardware setzen Geräte-Treiber und Dispatcher auf. Die Aufgabe eines Dispatchers ist die Zuteilung der Prozesse an den Scheduler. Er kann Prozesse entziehen und übergeben. In welcher Reihenfolge die Prozesse jedoch abgearbeitet werden, entscheidet der Scheduler, der mit dem virtuellen Speicher auf der nächst höheren Schicht implementiert ist. Darüber liegt das Dateisystem, das das Betriebssystem unterstützt. Die Interprozesskommunikation liegt in selbiger Schicht. Neben der Komponente für System-Aufrufe der Anwendungsprogramme, existiert auf dieser Abstraktionsschicht das Virtual File System (VFS, Virtuelles Datei-System). Es ist eine Abstraktionsschicht, die oberhalb des Dateisystems angeordnet ist. Sie bietet den Anwendungsprogrammen eine einheitliche API an, um auf unterschiedliche

Dateisysteme zugreifen zu können. Dadurch ist es möglich, auch auf betriebssystemfremde Dateiformate zuzugreifen.

Ein **Vorteil** dieser Architektur ist der Performance-Gewinn durch die einheitliche Implementierung im Kernel-Modus. Dadurch entfallen rechenaufwendige Wechsel zwischen Kernel- und Benutzer-Modus im Betriebssystem. Zudem entfallen einige Kommunikationsfunktionen.

Der privilegierte Zugriff aller Komponenten auf die Hardware im Kernel-Modus kann sich als **Nachteil** herausstellen. Wird eine effiziente Modularisierung verpasst, könnte der Absturz einer Komponente zum Absturz des gesamten Systems führen. Zudem wäre es schwierig Systemkomponenten auszuwechseln. Diese Nachteile können durch ein modularisiert gestaltetes Betriebssystem abgeschwächt bzw. vermieden werden. Die Modularisierung ist dann Aufgabe des entsprechenden Betriebssystems und die der Architektur.

Als Betriebssysteme, die einen Monolithischen Kernel besitzen können Linux und UNIX genannt werden. Die Umgebung des Dateisystems in Linux wird in Kapitel 4 beschrieben.

3.3 Hybrid-Kernel

Der Hybridkernel ist eine variable Architektur zwischen Mikrokernel und einem Monolithischen System. Welche Systemkomponenten im Benutzer- bzw. Kernel-Modus implementiert werden, liegt an der individuellen Architektur eines Betriebssystems. Dieser Kernel versucht Vorteile des Monolithischen- und Mikro-Kernels zu verbinden. Windows NT bis Windows 7 und Mac OS X sind Betriebssysteme, die nach dieser Architektur konzipiert sind.

4 Fallbeispiel Betriebssystem Linux und MINIX

In diesem Kapitel wird die Architektur des Betriebssystems Linux vorgestellt. Der allgemeinen Erläuterung zum Kernelaufbau folgt ein Einblick in die nähere Umgebung oberhalb der lokalen Dateisysteme. Abschließend folgt eine kurze Beschreibung des MINIX-Betriebssystems, das mit einem Mikrokernel arbeitet.

4.1 Der Architekturaufbau des Betriebssystemkerns von Linux

Im Kapitel 3.1 wurde der Idealtyp der Betriebssystemarchitektur des Mikrokernels ausführlich beschrieben. Dieser zeigt einen modularen Aufbau der Komponenten und eine Trennung zwischen Kernel- und Benutzer-Modus. Ein Vorteil ist der einfache Austausch einer Komponente, ein anderer die Stabilität des Systems bei Ausfall einer einzelnen Komponente. Ein Nachteil hingegen ist der Performanceverlust durch Wechsel zwischen Kernel- und Benutzermodus. Diese Vorteile soll Linux aufgreifen, deren Nachteile verhindern.

Linus Benedict Torvalds ist Initiator des Linux-Kernels, den er ursprünglich geschrieben hatte, um seinen Computer besser zu verstehen und ein UNIX-System als Vorlage besaß. Linux besaß ursprünglich einen stark modularisierten reinen monolithischen Betriebssystemkern. Dadurch nutzte Linux die Vorteile der schnellen Performance eines Monolithischens Kernels und begegnete den Nachteilen der Stabilität bei Ausfall einer Betriebssystemkomponente durch strikte Modularisierung innerhalb des Betriebssystems. Der Ausfall einer Systemkomponente führt daher nicht zum kompletten Absturz des Systems. Zudem ist es möglich während der Nutzung Module zu- oder abzuschalten. Dies ermöglicht Linux theoretisch die Konfigurationsmöglichkeit eines minimalen Systems. Weitere Komponenten können während der Laufzeit dynamisch zugeschaltet werden. Die Ausnahme bilden für das System essenzielle Module.

Auch wenn heute zunehmend Module im Benutzer-Modus (z.B. FUSE) zugeschaltet werden, wird Linux weiterhin als Beispiel für ein Monolitsches System angeführt. Die Abbildung 4.1 zeigt die Architektur von Linux mit „reinem“ Monolitischen Kernel. Die in Kapitel 3.1 und 3.2 vorgestellten Komponenten sind in der Grafik weiß hinterlegt. Dieses Kapitel wird sich themen- und rahmenbedingt auf die Erklärung der Speicherkompontenten (storage) beschränken.

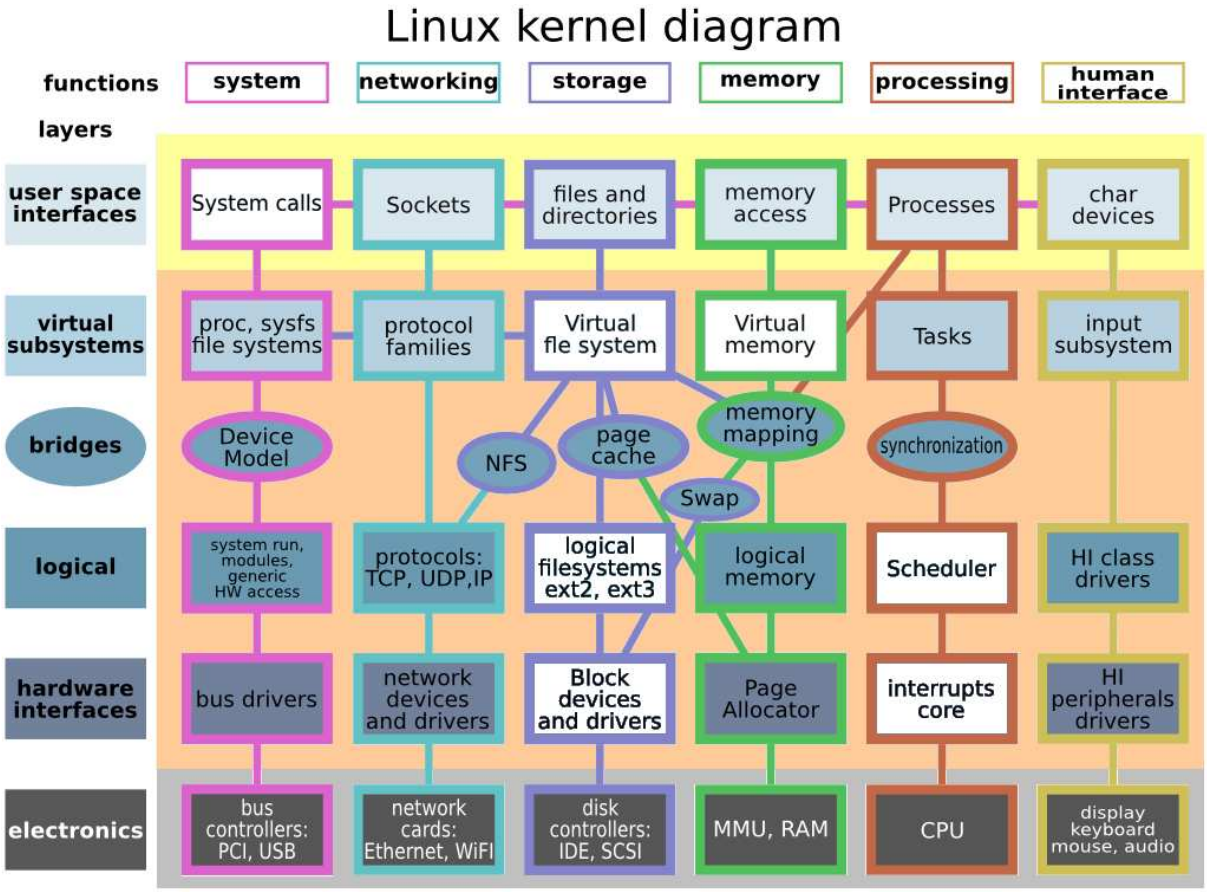

Abbildung 4.1

Aufbauend auf der Speicher-Hardware liegt die Schicht mit den **Gerätetreibern** (Block Devices and Drivers), die es Linux ermöglichen, über diese Schnittstelle auf die (auf Speicherblöcken basierte) Hardware zuzugreifen. Die nächsthöhere Abstraktionsschicht stellt das **logische Dateisystem** zur Verfügung. Linux nutzt seit Ende 2008 das Format ext4. Weitere ältere Dateisysteme (ext2, ext3) können unter Linux parallel genutzt werden. Oberhalb der logischen Dateisysteme sorgt das **Virtual Filesystem** (als letzte Schicht im Kernel-Modus) für eine einheitliche Schnittstelle zu den Verzeichnissen und Dateien, unabhängig von dem verwendeten Dateisystem. Zwischen den logischen und virtuellen Dateisystemen fungieren **NFS** als Brücke vom virtuellen Speichersystem zum Netzwerk und **Swap** als Brücke zum Arbeitsspeicher. Der Swap kann ganze Prozesse zwischen Platte und Speicher verschieben, wenn der physische Speicher für aktive Prozesse nicht mehr ausreicht. Es ist zudem auch möglich, nur Teile der Datei zu verschieben, so dass nur noch Seitentabellen/ Benutzerstruktur eingelagert ist. Der Prozess gilt dann trotzdem als „im Speicher“. Text-, Daten-, Stacksegement wird bei Bedarf dynamisch angefordert. Der **Page Cache** sowie weitere Caches werden in Kapitel 4.2 beschrieben.

4.2 Die Kernel-Umgebung des Linux-Dateisystems

Da sich diese Arbeit primär auf Betriebssystemschichten im Rahmen von Dateisystemen bezieht, wird dieser Abschnitt die nähere Umgebung des Dateisystems (lokal oder verteilt) beschreiben. Hierbei wird der Fokus auf die Module oberhalb der physischen Dateisysteme liegen.

Dateisysteme besitzen alle ähnliche Funktionalitäten und Beschränkungen. Auch in Linux können z.B. Verzeichnisse absolut oder relativ (vom Arbeitsverzeichnis ausgehend) angesprochen werden, es gibt dateisystemabhängige Namenlängen/ -zeichen für Dateien, jedoch keinen Zwang für Dateiendungen (unabhängig vom Zwang für Dateiendungen durch Anwendungen).

Ein markanter Unterschied ist jedoch, dass das Dateisystem von Linux einer „Stammwurzel“ entspringt. Dem „Stamm-Dateisystem“ des Betriebssystems werden andere Dateisysteme einfach an einer beliebigen Stelle angehängt. Dabei ist es für Linux unbedeutend, ob es sich bei den Dateisystemen um Festplatten, DVD-Laufwerke, Speicherkarten, Netzwerk-dateisysteme oder sonstige Medien handelt. Siehe hier Abbildung 4.2.1.

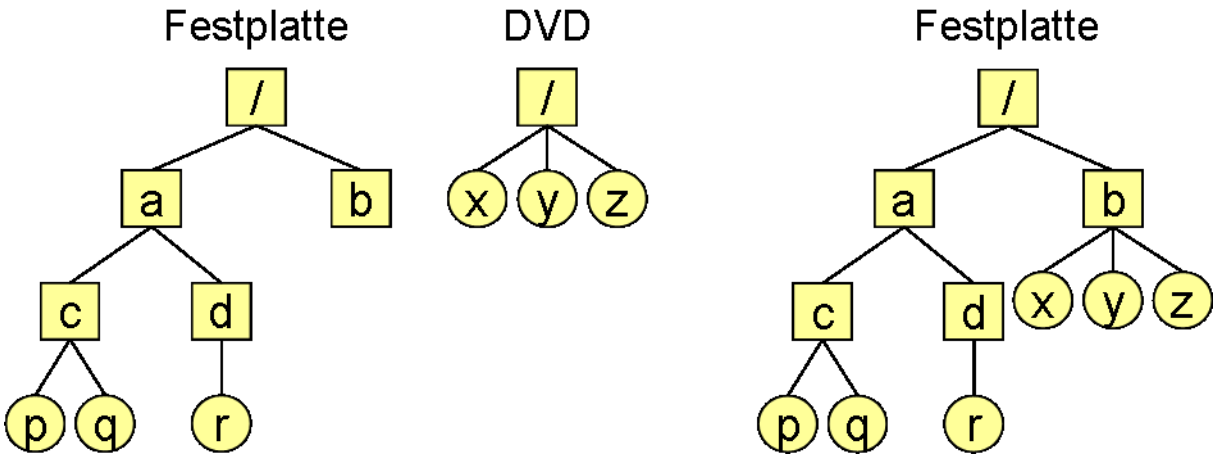

Abbildung 4.2.1

Anmerkung: In der einführenden Literatur zum Virtuellen Dateisystem von Linux werden die Begriffe Virtual Filesystem und Virtual Filesystem Switch oft nicht differenziert. Auch diese Arbeit nutzt diese Begriffe synonym. Das **Virtual Filesystem** bzw. der **Virtual Filesystem Switch** (VFS, virtuelles Dateisystem) bietet eine einheitliche Schnittstelle für unterschiedliche Dateisysteme und baut auf dem POSIX-Standard auf. Dieser definiert, welche Operationen Dateisysteme einem virtuellen Dateisystem anbieten müssen. Werden nicht alle Funktionen angeboten, müssen Wrapper unterhalb des VFS die fehlenden bereitstellen.

Die Funktionen sind in Dateisystemstrukturen enthalten. Dabei gehören Superblock, I-Node, Dentry und File zu den wichtigen Strukturen. Der Superblock enthält kritische Informationen über den Aufbau eines Dateisystems und enthält z.B. Zeiger auf die I-Nodes, freie Blöcke und Größe des Dateisystems. I-Node ist die Abkürzung für Index-Node (Index-Knoten). Unter Angabe dieser Nummer identifiziert ein Dateisystem die Lage einer Datei auf dem Datenspeicher. Zudem werden auch Geräte und Verzeichnisse mittels I-Nodes verwaltet. Dentry-Einträge repräsentieren „harte“ Pfadverläufe wie z.B. /usr/ast/bin. Werden Pfade mehrfach durch Prozesse genutzt, vermeiden diese Abbildungen doppelte Verzeichnisangaben und –suchen (die sonst immer erneut linear von I-Node (usr/) zu I-Node (ast/…) erfolgen würde). Die File-Datenstruktur ist eine speicherinterne Darstellung einer geöffneten Datei. Die jedem dieser Systemstrukturen zugeordneten Operationen sind Zeiger auf Funktionen der zugrunde liegenden Dateisysteme, die nicht syntaktisch, jedoch semantisch gleich sein müssen.

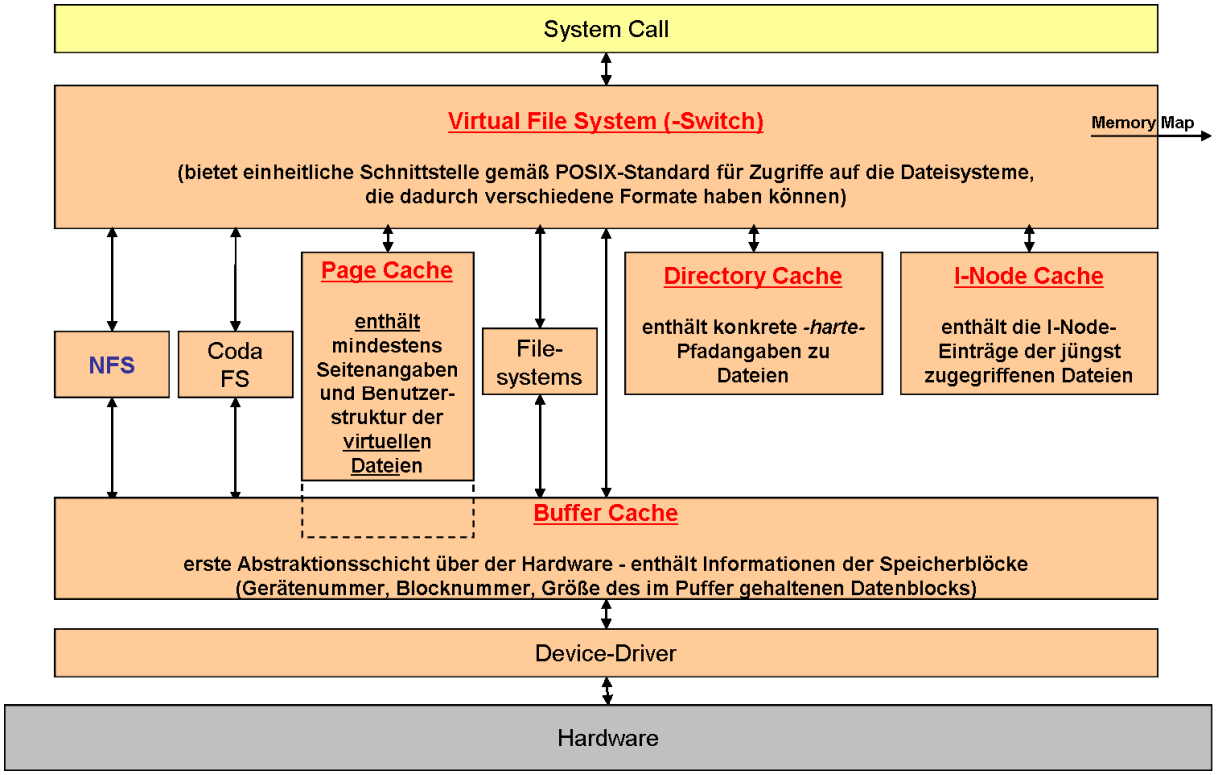

Abbildung 4.2.2

In **Caches** werden genutzte Inhalte für einen erneuten Zugriff gehalten. Sie sind im Speicher implementiert und bieten dadurch Performancevorteile gegenüber dem Zugriff von einer Festplatte. In Linux werden mehrere Caches genutzt. Abbildung 4.2.2 zeigt eine schematische Übersicht. Coda FS ist ein im Netzwerk verteiltes Dateisystem. Es wird in dieser Arbeit nicht weitergehend beschrieben, sondern soll noch einmal die Flexibilität der mit dem VFS zu verwaltenden Dateisystemen und Netzwerkausrichtung (besonders von Linux) unterstreichen.

Der Buffer Cache ist die erste Abstraktionsschicht über der Hardware und enthält Informationen der Speicherblöcke (Gerätenummer, Blocknummer, Größe des im Puffer gehaltenen Datenblocks). Der Page Cache enthält mindestens Seitenangaben und Benutzerstruktur der virtuellen Dateien (siehe Beschreibung Swap im vorherigen Kapitel). Seit dem Kernel 2.4 sind Page Cache und Buffer Cache zusammengelegt. Eine Abfrage nach dem Inhalt durch „free –m“ weist jedoch auch weiterhin beide Caches getrennt aus. Für die eingangs beschriebenen Dentry-Einträge existieren Directory Caches und für die I-Nodes ist ein I-Node Caches vorhanden. Alle Caches sind in Linux i.d.R. nach dem Algorithmus

LRU (Least Recently Used) implementiert, bei dem der am längsten nicht verwendete Block ausgelagert bzw. aus dem Cache gelöscht werden.

Wie bereits beschrieben können die verbundenen Dateisysteme u.a. als Festplatte oder als optisches Laufwerk, lokal oder im Netzwerk vorliegen. Zwischen ihnen und dem VFS ermöglicht das **Network File System** (**NFS**, Netzwerk Dateisystem) die Kommunikation zu den entsprechenden Dateisystemen. Über das NFS werden Verbindungen zu Verzeichnissen und Dateien verwaltet, die sich im Netzwerk befinden (LAN oder WAN). Die Verbindung kann beim Hochfahren des PC´s oder erst per Automounting bei Zugriff hergestellt werden. Für Lesezugriffe bietet Automounting die Möglichkeit, auf eine Datei zugreifen zu können, die redundant auf mehreren Servern platziert ist. Dies hat natürlich Performancevorteile, denn statt dem evtl. schlechten Verbindungsaufbau zu einem bestimmten Server, kann der nächstmöglich erreichbare angesprochen werden. Das VFS enthält virtuelle I-Nodes (V-Nodes). Diese verweisen auf lokale I-Nodes oder entfernte I-Nodes (Remote I-Nodes bzw. R-Nodes) innerhalb von NFS-Clients, die wiederum Anforderungen an im Netzwerk befindliche Dateien Stellen. Dieser Ablauf wird in Abbildung 4.2.3 dargestellt.

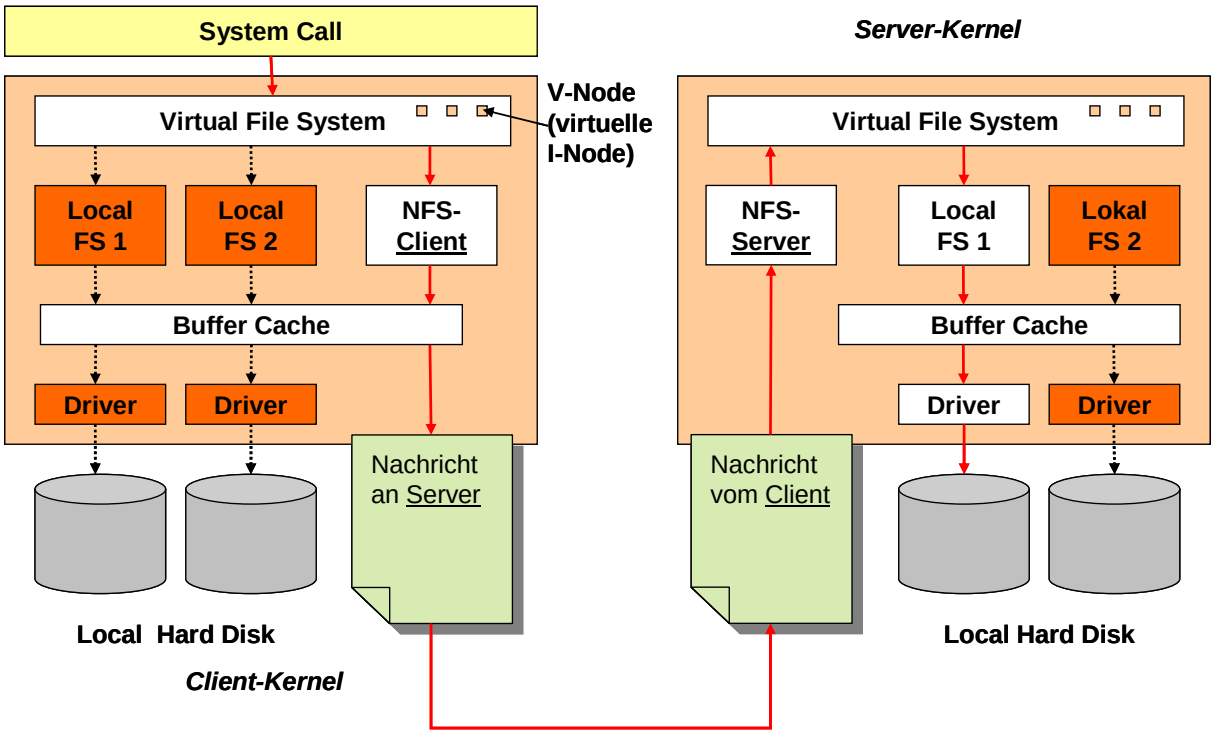

Abbildung 4.2.3

Die nächst niedrigere Abstraktionsebene ist der physische lokale Datenspeicher. Wie oben beschrieben, ist es durch das VFS möglich, unterschiedlich formatierte Datensysteme anzusprechen. Wie UNIX nutzte auch Linux anfänglich MINIX-1, danach jedoch **extended filesytems** (derzeit **ext 4**). Zur Erklärung von Aufbau und Funktionen wird z.B. auf Tanenbaum „Moderne Betriebssyteme, 3. aktualisierte Auflage“ 2009 verwiesen. Die unterste Abstraktionsschicht eines Betriebssystems schließt mit **Gerätetreibern** ab, die direkt auf die Hardware zugreifen.

4.3 MINIX - Ein auf einem Mikrokernel basierendes Betriebssystem

Nach der Beschreibung des Betriebssystems Linux mit Monolithischem Kernel stellt dieses Kapitel mit Minix ein Betriebssystem vor, das einen Mikrokernel besitzt. Minix entstand durch Modularisierung und Re-Implementierung eines UNIX-Systems mit dem Ziel ein einfaches Mikrokernelsystem zu Lehrzwecken anzubieten. Urvater dieses Systems ist Andrew S. Tanenbaum. Die Betriebssytemkomponenten von MINIX bieten größtenteils Funktionen an, die in vorangegangenen Kapiteln beschrieben wurden. Vor- und Nachteile lassen sich dem Kapitel 3 entnehmen. Daher soll dieses Kapitel lediglich durch Nennung und Einordnung der Komponenten einen Kontrast zu dem vorher beschriebenen Systems darstellen.

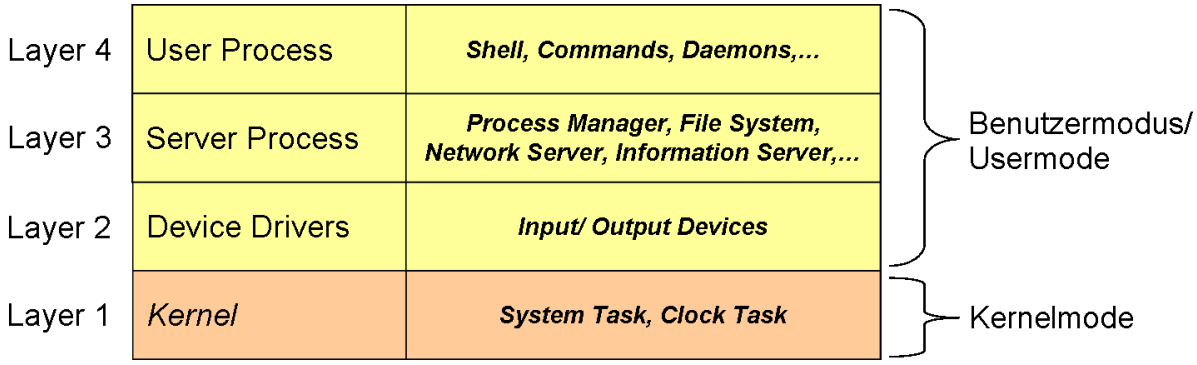

Abbildung 4.3

Die Abbildung 4.3 zeigt den Aufbau von Minix über 4 Schichten. Jede Schicht bietet der nächst höheren Schicht Funktionen an. Dabei läuft lediglich Schicht 1 im Kernel-Modus und enthält u.a. Scheduler, Dispatcher, IPC. In Schicht 2 sind die Gerätetreiber angeordert. Schicht 3 bietet alle Services an, die Programme aus Schicht 4 benötigen. Hierzu zählen natürlich File System, Process Manager, Network Server oder Memory Manager. In Schicht 4 befinden sich Shell, Daemons und Anwendungsprogramme des Benutzer, die Funktionen der darunter liegenden Schichten verwenden.

5 Zusammenfassung

Was steckt hinter der Benutzeroberfläche von Betriebssystemen? Fragen wie dieser soll diese Arbeit Antworten geben. Dabei liegt der Fokus auch auf die Umgebung der Dateisysteme.

Betriebssysteme sind modular oder in Schichten aufgebaut. Zum Betriebssystem gehört jede Software, die Anwendersoftware das Arbeiten mit der zugrunde liegenden Hardware ermöglicht. Die Zugriffe können im Kernel-Modus mit vollem Zugriff oder im Benutzer-Modus mit eingeschränktem Zugriff auf die Hardware erfolgen. Das Betriebssystem legt fest, in welchem Modus ihre Komponenten auf die Hardware zugreifen. Zwei extreme Idealtypen stellen Mikro- und Monolithischer Kernel dar. Bei der Mikrokernelarchitektur arbeiten lediglich die für das Betriebssystem essenzielle Komponenten im Kernel-Modus, alle anderen im Benutzer-Modus. Demgegenüber arbeiten bei einem Monolithischen Kernel alle Komponenten im Kernel-Modus. Markante Faktoren bei Betriebssystemen sind Performance und Stabilität. Mikrokernel haben grundsätzlich den Vorteil bei der Stabilität, da ein Absturz vereinzelter nicht essenzieller Komponenten nicht das ganze Betriebssystem zum Absturz bring. Einen Nachteil bringt der Performanceverlust durch Wechsel zwischen Kernel- und Benutzer-Modus mit sich. Hybrid-Kernel sind eine Mischform dieser beiden Architekturen.

Das Betriebssystem Linux arbeitet grundsätzlich mit einem Monolithischen Kernel. Die Betriebssystemkomponenten sind modular aufgebaut, so dass deren Ausfall sich nicht auf die Lauffähigkeit des Betriebssystems auswirken soll. Durch Modularisierung soll die Stabilität unterstützt werden, während die Performance-Vorteile architekturbedingt durch den Monolithischen Kernel wirken. Allerdings gibt es bei Linux Entwicklungen, die von einem idealtypischen Monolithischen Kernel abweichen. Wie genannt, arbeitet z.B. das Modul FUSE im Benutzer-Modus. Page Cache/ Buffer Cache, Directory Cache und I-Node Cache steigern zudem die Performance beim erneuten Zugriff auf Dateien oder Verzeichnisse. Um weiteren Stellvertretern des Betriebssystemmarktes gerecht zu werden, soll angemerkt sein, dass es sich bei den meisten Mac OS X und Windows-Versionen um Hybrid-Kernel handelt, auch wenn sich der Windows 7 bzw. Vista-Kernel stark dem Monolithischen Schema annähert.

In dieser Arbeit wurde ein Ausschnitt existierender Konzepte im Aufbau von Betriebssystemen beschrieben. Wie in anderen Bereichen der Softwarewelt lässt sich kaum ein Urteil aus gut oder schlecht bilden, sondern viel mehr aus geeignet oder ungeeignet. Anforderungen und technischer Stand wirken sehr stark auf die Eignung einer Software (z. B. stellt eine Mehrbenutzersoftware andere Anforderungen an Stabilität und Konsistenz der Daten, als eine Bildbearbeitungssoftware). Mit neuen Modulen entwickelt sich Linux derzeit zu einem hybriden Betriebssystem. Aktuelle Entwicklungen der Technik und Betriebssystemkomponenten sowie deren Implementation und neue Anforderungen an Sicherheit sowie Stabilität mögen diese Entwicklung (für den Moment) bestätigen.

6 Abbildungsübersicht

7 Quellenverzeichnis

Bücher:

Andrew S. Tanenbaum „Moderne Betriebssyteme, 3. aktualisierte Auflage“

Internet:

http://kris.koehntopp.de/artikel/diplom/node21.html

http://www.linux-tutorial.info/modules.php?name=MContent&pageid=310

http://www.ibm.com/developerworks/linux/library/l-virtual-filesystem-switch

http://de.wikipedia.org/wiki/Betriebssystemkern

http://de.wikipedia.org/wiki/Mikrokernel

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

http://de.wikipedia.org/wiki/Hybridkernel

http://de.wikipedia.org/wiki/Exokernel

http://de.wikipedia.org/wiki/Symmetrisches_Multiprozessorsystem

http://de.wikipedia.org/wiki/Linux_(Kernel)

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

http://de.wikipedia.org/wiki/Interprozesskommunikation

http://de.wikipedia.org/wiki/Virtuelle_Speicherverwaltung

http://de.wikipedia.org/wiki/Dispatcher

http://de.wikipedia.org/wiki/Virtuelles_Dateisystem

http://www.thomas-krenn.com/de/wiki/Linux_Page_Cache_Grundlagen

http://imma.wordpress.com/2007/04/02/presentation-internal-structure-of-minix/

http://de.wikipedia.org/wiki/Minix_%28Betriebssystem%29