Betriebssysteme

Hans Jürgen Ohlbach

27. November 2017

Keywords:

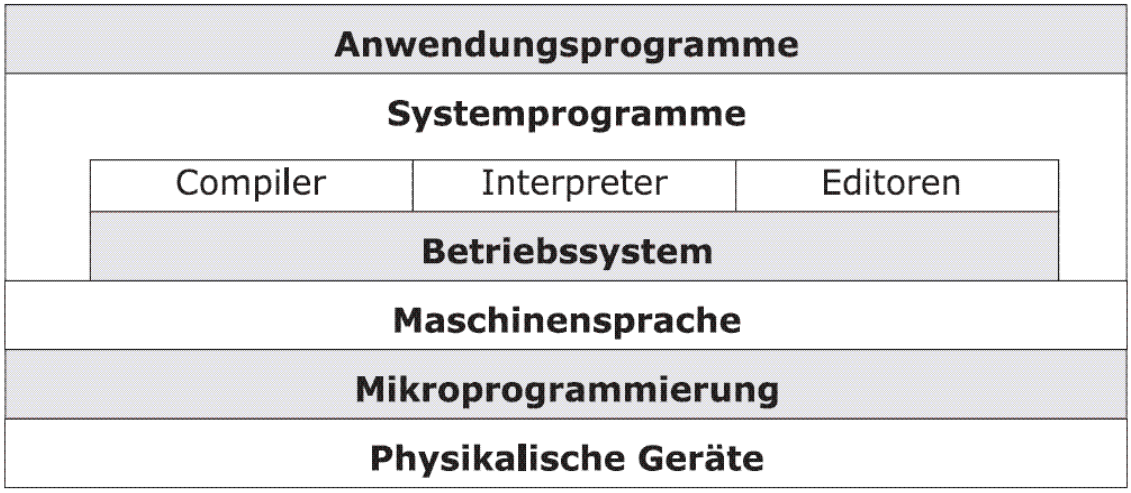
Inhaltsverzeichnis

Dieses Miniskript gibt einen ersten groben Überblick über das Thema Betriebssysteme. Die einzelnen Komponenten werden in den entsprechenden Miniskripten besprochen.

1 Überblick

Jeder moderne Staat, der nicht im Chaos enden will, braucht Gesetze und eine öffentliche Verwaltung. Sie bilden den Rahmen, in dem die einzelnen Akteure, Personen, Firmen, Vereine usw. agieren. Ganz ähnlich ist es auch mit Computern. Sie haben ja nicht nur einen Prozessor, sondern auch eine Vielzahl anderer Geräte: Arbeitsspeicher, Festplatte, Monitor, Keyboard, externe Anschlüsse usw. die in sinnvoller Weise miteinander agieren, und dem Benutzer in möglichst einfacher Weise zur Verfügung stehen sollen. Damit das reibungslos funktioniert, dafür sorgt das *Betriebssystem* eines Computers.

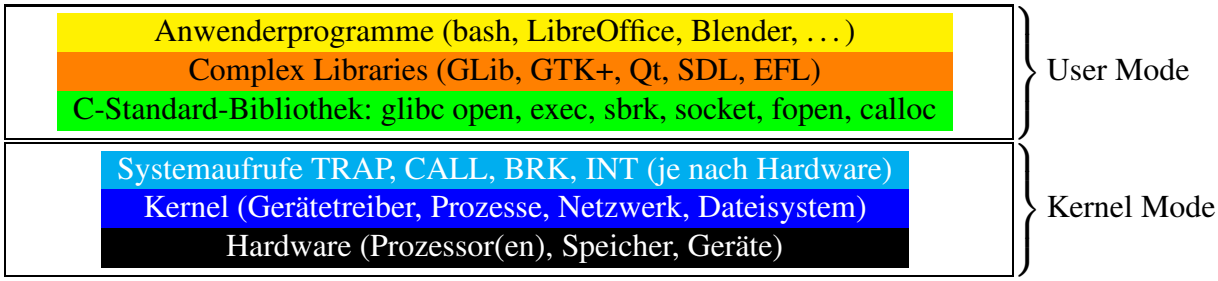
In jedem Computer gibt es eine Hierarchie von Abstraktionsstufen, die in etwa folgendem Bild entspricht:



Ganz unten in der Hierarchie ist die Hardware mit Prozessor, Speicher, Festplatte, Monitor, IO-Kanäle usw. Einige der Geräte, insbesondere Prozessoren, lassen sich an bestimmte Anforderungen anpassen, indem einzelnen Maschinenbefehle per *Mikroprogrammierung* definiert werden. Nach außen hin hat jeder Prozessor eine festgelegte *Maschinensprache*, mit der er programmiert wird.

In der Mitte der Hierarchie liegt dann das Betriebssystem, welches alle Abläufe innerhalb des Computers steuert. Mit dem Betriebssystem kommen meist auch eine Reihe von Systemprogrammen, wie Compiler, Interpreter, Editoren usw. die aber nicht unbedingt Teil des eigentlichen Betriebssystems sind, sondern eher Entwicklungswerkzeuge. Die oberste Schicht schließlich bilden die Anwendungsprogramme.

Das folgende Bild gibt einen etwas genaueren Eindruck davon am Beispiel von Linux:



Hier sieht man auch die zwei wichtigen sog. *Modes*: *Kernel Mode* und *User Mode*. Im Kernel Mode laufen die sicherheitskritischen Komponenten des Betriebssystems. Sie brauchen eine spezielle Berechtigung, eben den Kernel Mode. Will z.B. ein Benutzerprogramm auf eine Datei zugreifen, dann muss überprüft werden, ob der Benutzer die Berechtigung dazu hat. Ein Dateizugriff ist erst nach Erteilung der Berechtigung dazu erlaubt. Technisch geht das so, dass das Anwendungsprogramm im User Mode einen sog. Systemaufruf macht. Der Systemaufruf überprüft die Berechtigung. Wenn sie erteilt wird, dann schaltet er in den Kernel Mode um, so dass alle weiteren Maschinenbefehle, die den Zugriff durchführen, erlaubt sind. Man geht dann davon aus, dass diese Programmteile fehlerfrei implementiert sind (zumindest hofft man das).

Da die Betriebssystemteile im Kernel Mode sehr sicherheitskritisch sind, und daher besonders sorgfältig implementiert und getestet werden müssen, versucht man, diese Teile so klein wie möglich zu halten. Ist der Teil sehr klein, spricht man auch von *Microkernel*.

Der Betriebssystem Zoo

Allerdings gibt es nicht *das* Betriebssystem, sondern einen riesigen Zoo von Betriebssystemen. Die Website http://www.operating-system.org/betriebssystem/_english/os-liste.htm listet (2017)

- 656 Linux-basierte Betriebssysteme,
- 611 nicht Linux-basierte Betriebssysteme auf.

Bei den Endanwendern werden allerdings nur ganz wenige davon benutzt. Das Unternehmen StatCounter analysiert die Verbreitung von Endanwender-Betriebssystemen anhand von Zugriffsstatistiken diverser Websites. Die laut StatCounter am weitesten verbreiteten Endanwender-Betriebssysteme in 2017 sind:

Windows	mit ca. 40 %
Android	mit ca. 40 %
iOS	mit ca. 15 %
macOS	mit ca. 5 %
Linux	mit ca. 1 %

Da es inzwischen die unterschiedlichsten Arten von Computern gibt, gibt es auch dafür optimierte Betriebssysteme. Man unterschiedet:

- **Mainframe-Betriebssysteme**: sie sind optimiert auf eine Vielzahl gleichzeitig ablaufender Prozesse mit zahlreichen hochperformanten I/O-Operationen auf einem sehr großen Hintergrundspeicher (z.B. große Webserver). Bsp.: OS/390, ein Nachfolger von OS/360,
- **Server-Betriebssysteme**: z.B. Unix oder Windows 2000, für kleine Webserver und Workstations.
- **Multiprozessor-Betriebssystem**: für Computersysteme mit mehreren Prozessoren,
- **PC-Betriebssysteme**: z.B.Windows 10, Mac OS X oder Linux. Als möglichst generische Plattform für eine Vielzahl von Anwendungen und Anforderungen,
- **Tablet-Betriebssysteme**: Android, iOS, Windows,...
- **Echtzeit-Betriebssysteme**: z.B.für Steuerung von Maschinen. Die Reaktion des Betriebssystems auf eine Steuerungsanweisung muss „auf den Punkt“ erfolgen. Je nach Toleranz spricht man von weichen bzw. harten Echtzeitsystemen,
- **Embedded Betriebssysteme**: für mobile Endgeräten (PDA). Ihre Charakteristiken sind: Effizientes Energiemanagement, eine Vielzahl von Kommunikationsschnittstellen, wie z.B. der Aufbau oder das Einloggen in ein Ad-Hoc-Piconetz (Bluetooth) oder die Einwahl in ein Mobilfunknetz (GSM, UMTS).
- **Betriebssysteme für Chipkarten**: Diese Mini-Betriebssysteme laufen auf Smart Cards, wie sie z.B. im Zusammenhang mit der RFID-Technologie eingesetzt werden. Sie basieren z.T. auf der Java Virtual Machine.

Wegen dieser Vielzahl können wir uns in diesem und allen dazugehörigen Miniskripten nur auf die grundlegenden Techniken und Prinzipien von Betriebssystemen konzentrieren.

2 Wichtige Komponenten eines Betriebssystems

Wir skizzieren die einzelnen Komponenten „von unten nach oben“. Ganz unten in der Abstraktionshierarchie ist die Hardware.

2.1 Gerätetreiber

Die Schnittstelle zur Hardware bilden die *Gerätetreiber*. Sie sind Softwarekomponenten, die dem Prozessor ermöglichen, die angeschlossenen Geräte zu steuern: Arbeitsspeicher, Festplatten, Drucker, Netzwerkkarte, Graphikkarte, Soundkarte, Scanner, digitale Kamera, USB-Schnittstelle, WiFi- und Bluetooth-Verbindungen usw.

Ein Gerätetreiber hat zwei Schnittstellen: die Schnittstelle zu dem Gerät selbst, indem es Steuersignale, z.B. über einen *Kommunikationsbus* an das Gerät schickt und Signale vom Gerät empfängt, sowie die Schnittstelle zum Betriebssystem. Da ein Gerätetreiber auch ein Programm ist, welches

vom Prozessor abgearbeitet wird, besteht die Schnittstelle zum Betriebssystem aus der Vereinbarung, wo und wie die Daten auszutauschen sind. D.h. der Prozessor speichert die Befehle und Daten für die Kontrolle eines Gerätes an bestimmten Stellen und ruft dann den Gerätetreiber auf. Dieser holt sich dort die Informationen, steuert das Gerät an, und legt die vom Gerät zurückgeschickten Daten wieder ab an einer Stelle, wo sie der Prozessor abholen und weiterverarbeiten kann.

Dazu werden i.A. bestimmte Adressbereiche im Arbeitsspeicher für das Gerät reserviert. Der Prozessor schreibt in diesen Bereich seine Befehle und Daten, und der Gerätetreiber liest sie dort aus und schreibt seine Daten auch in diesen Bereich. Es kann auch spezielle *Device Control Register* für den Datenaustausch mit dem Treiber geben.

Meist arbeiten der Prozessor und das Gerät parallel. Ist das Gerät fertig, muss es das dem Prozessor mitteilen. Dazu kann es *Interruptsignale* schicken. Ein Interruptsignal bewirkt, dass der Prozessor bei seiner momentanen Arbeit unterbrochen wird, und ein bestimmtes Programm, den *Interrupt Handler* aufruft. Ist der Interrupt Handler fertig, kehrt das Programm wieder an die Stelle zurück, wo es vorher war.

Direct Memory Access
Ein Gerätetreiber steht in Konkurrenz zu den anderen Prozessen im Betriebssystem. Wenn er arbeitet, blockiert er den Prozessor für die anderen Prozesse. Um z.B. eine große Datei von der Festplatte in den Hauptspeicher zu laden, werden die Daten wortweise (32 oder 64 Bits) in die Register, und von da in den Arbeitsspeicher kopiert. Das kann sehr lange dauern.

Da dies sehr oft benötigt wird, hat man ein Art Abkürzung entwickelt: den DMA-Chip (Direct Memory Access). Wenn der Prozessor jetzt eine Datei laden will, dann genügt es, dem DMA-Chip mitzuteilen, welche Datei geladen werden soll, und wohin die Daten in den Arbeitsspeicher kopiert werden sollen. Ab diesem Moment können Prozessor und DMA-Chip parallel arbeiten. Der DMA-Chip kommuniziert mit der Festplatte und sorgt dafür, dass die Daten an die richtige Stelle im Arbeitsspeicher kommen. Ist die Übertragung beendet, schickt der DMA-Chip dem Prozessor ein Interruptsignal, so dass der Prozessor weiß, dass die Daten da sind. Das hat die Kommunikation mit externen Datenspeichern erheblich beschleunigt.

Wenn man sich in den verschiedenen Programmiersprachen die Lesebefehle auf der niedrigen Ebene der Eingabeströmen ansieht (in Java z.B. InputStream.read(byte[])), dann sind die immer so, dass man vorher ein Array einer festen Länge anlegen muss, in das die Daten hineingeschrieben werden. Dies ist genau der Speicherbereich, in den der DMA-Chip schreibt.

3 Prozesse

Ein Prozess ist ein „in Abarbeitung befindliches Programm“. Die Verwaltung der Prozesse ist eine der ganz wesentliche Kernaufgaben eines Betriebssystems. Dazu gehört insbesondere:

- das Laden eines Programms und die Bildung eines Prozesses,
- die Speicherverwaltung für den Prozess,
- die Auslagerung von Prozessen in den *Swapspace* auf der Festplatte, falls der Arbeitsspeicher zu voll ist,
- die parallele oder pseudo-parallele[1] Abarbeitung mehrerer Prozesse,
- die gemeinsame Nutzung von Bibliotheken durch mehrere Prozesse,
- die Verwaltung von Verbindungen zur Peripherie (offene Dateien, Sockets, Pipes usw.).

Der Bildung eines Prozesses gehen aber noch verschiedene Schritte voraus:

- Ein *Compiler* übersetzt den Quellcode von Programmteilen in Maschinencode.
- Ein *Linker* bindet verschiedene übersetzte Programmteile und evtl. statische Bibliotheken zu einem lauffähigen *Objektfile* zusammen.

Einen solchen Objektfile kann dann der *Lader* noch mit *dynamischen Bibliotheken* verknüpfen, und daraus eine Prozess formen.

Wie das alles im Einzelnen geschieht, wird in den Miniskripten *Prozesse* und *Prozesserzeugung* beschrieben.

4 Threads

In modernen Computern laufen viele Prozesse parallel oder pseudo-parallel. Aber auch innerhalb eines Prozesses kann es parallele Abläufe geben, sog. *Threads*. Ein typisches Beispiel ist ein Serverprogramm, welches viele Clients bedienen soll. Für jeden Client wird ein eigener Thread erzeugt, der parallel zu den anderen Clients und zu dem Hauptprogramm läuft. Damit wir erreicht, oder zumindest der Eindruck geweckt, dass alle Clients echt parallel bedient werden.

In früheren Betriebssystemen musste dieses Verhalten rein auf der Anwendungsebene programmiert werden. Die neueren Betriebssysteme unterstützen das jedoch direkt, was zu einer einfacheren Programmierung und zu mehr Effizienz geführt hat.

Im Miniskript *Threads* wird das ausführlicher erläutert.

[1] Pseudo-parallel bedeutet, dass sich mehrere Prozesse einen Prozessor teilen, indem sie sich so schnell abwechseln, dass der Eindruck echter Parallelität entsteht.

5 Filesysteme

Die Verwaltung von Filesystemen ist ebenfalls ein integraler Bestandteil von Betriebssystemen. Da der Zugriff auf Dateien sehr sicherheitskritisch ist, kann das nicht durch irgendwelche Anwenderprogramme selbst gemacht werden, sondern läuft über die Filesystem-Komponente des Betriebssystems.

Ein Filesystem ist zuständig für

- die Speicherung von Dateien und Verzeichnissen auf externen Speichermedien (Festplatte, SSD-Disk, USB-Stick usw.),
- die Verwaltung der Metadaten, wie z.B. Eigentümer, Zugriffsrechte , Zeitstempel usw.
- den kontrollierten Zugriff auf die Dateien,
- die Wiederverwendung von frei gegebenem Speicherplatz,
- Reparaturmaßnahmen, falls das Filesystem, z.B. durch Absturz des Rechners, inkonsistent geworden ist.

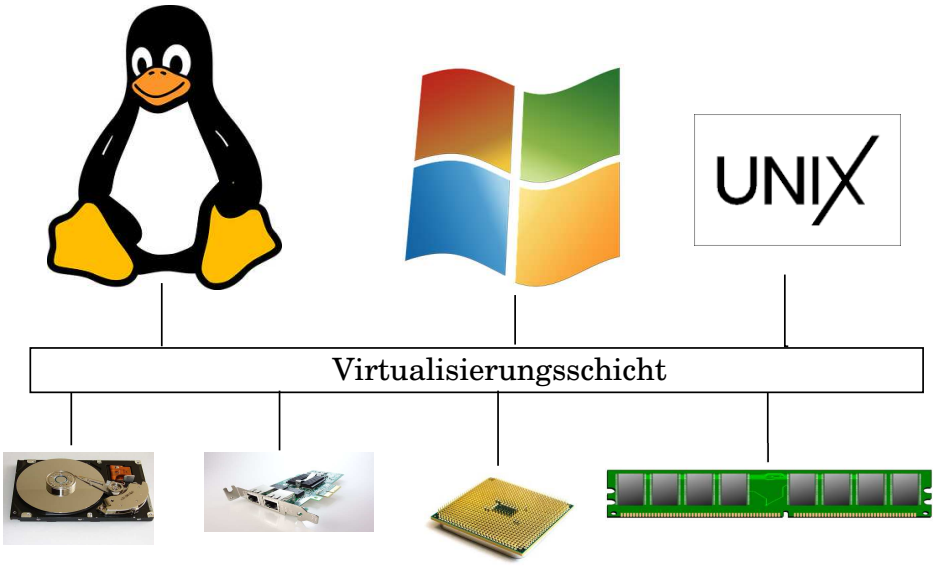
Aber auch hier gibt es nicht *das* Filesystem, sondern eine große Vielzahl davon. Sie werden permanent weiterentwickelt und neuen Anforderungen und Entwicklungen angepasst.

Das Miniskript *Filesysteme* gibt einen Überblick über die Ideen und Strukturen, die man in den meisten Filesystemen findet.

6 Virtualisierung

Die Grundidee bei der Virtualisierung ist folgende: Man hat zunächst ein Basissystem, z.B. einen konkreten Prozessor, ein Netzwerk, oder irgend ein anderes technisches System. Über dieses Basissystem legt man eine sog. *Virtualisierungsplattform*, die einer Anwendung ein ganz anderes System „vorgaukelt“. Die Anwendung arbeitet also nicht mit dem Basissystem, so wie es ist, sondern agiert so, als ob sie in einer ganz anderen Umgebung eingebettet wäre. Allerdings muss es nicht immer eine ganz andere Umgebung sein, die vorgegaukelt wird. Man spricht auch schon von Virtualisierung, wenn der Anwendung vorgegaukelt wird, sie sei ganz alleine und hätte das Basissystem allein für sich.

Im Bereich Betriebssysteme bedeutet es, einen Rechner so zu konfigurieren, dass mehrere Betriebssysteme, oder auch mehrere Instanzen des gleichen Systems *parallel* laufen können, ohne sich gegenseitig zu stören.



Warum sollte man so etwas wollen? Es gibt eine ganze Reihe von Gründen:

- Testen eines Programms für verschiedene Betriebssysteme,
- Testen neuer oder möglicherweise fehlerhafter Programme, die das Betriebssystem zerstören könnten (Schulungsumgebungen),
- Einführung zusätzlicher Dienste, die eventuell ein laufendes System stören könnten,
- Dienste, die alleine einen Rechner nicht auslasten,
- Abschottung verschiedener Dienste gegeneinander
  (Hacker können dann nur eines davon angreifen),
- jedem Benutzer die Möglichkeit schaffen, seine ganz individuelle Umgebung zu konfigurieren,
- Programme weiter zu nutzen, die nur auf veralteter oder nicht verfügbarer Hardware laufen.

Im Miniskript zur Virtualisierung werden dazu folgende Ansätze dazu vorgestellt:

Paravirtualisierung: ein Rechner wird in mehrere virtuelle Maschinen aufgeteilt, in denen verschiedene Betriebssysteme (für dieselbe Architektur) laufen.

Emulation: simulieren einen kompletten Rechner per Software (Beispiel: Java)

Partitionierung: ein Betriebssystemkern läuft nur einmal, stellt aber mehrere unabhängige Laufzeitumgebungen zur Verfügung.

API-Emulatoren: ein Betriebssystem A stellt die Bibliotheken eines Betriebssystems B zur Verfügung, so dass Programme von B laufen können in A.

Dynamisches Recompilieren: (Binary Translation) Zur Laufzeit wird der Code Block für Block in Code der Zielmaschine übersetzt.

Stichwortverzeichnis