1

Einführung

1.1 Was ist ein Betriebssystem?

Ein Versuch, den Begriff *Betriebssystem* zu definieren, scheitert sehr schnell. Wichtig sind jedoch die Beobachtungen, die wir bei diesem Versuch machen können:

- Auf ein und demselben Rechner können unterschiedliche Betriebssysteme laufen:
  So kann auf einem gängigen PC mit x86-Prozessor sowohl Linux – ein Unix-ähnliches Betriebssystem – als auch Windows eingesetzt werden.
- Auf Rechnern unterschiedlicher Hardware kann das gleiche Betriebssystem arbeiten:
  So kann Linux auf einem PC, aber auch auf leistungsstarken Servern diverser Hersteller implementiert sein.

Im ersten Falle erleben wir bei der Benutzung bei gleicher Hardware große Unterschiede, im zweiten Falle trotz unterschiedlicher Hardware nahezu gleiches Verhalten der Rechner.

Diese Beobachtung führt uns zu anderen Fragen:

- Wie hängen Hardware und Betriebssystem zusammen?
- Kann man einzelne Aufgaben eines Betriebssystems eingrenzen und beschreiben?
- Wie erleben wir als Benutzer ein Betriebssystem?

Die folgende Abb. zeigt den geschichteten Aufbau: die Hardware besteht aus Sicht des Betriebssystems zunächst aus der Menge der Maschineninstruktionen. Diese werden heute üblicherweise durch einen Mikrocode der CPU definiert und abgearbeitet. Dieser Mikrocode greift physisch auf die Schaltkreise zu. Die Schaltkreise können auch komplexe Geräte wie z.B. ein Platten-Controller sein.

Die Anwendungsprogramme sehen nicht die Hardware, sondern das Betriebssystem. Dieses stellt somit eine Erweiterung der Hardware dar, die einerseits den Anwendungsprogrammen die Mittel bereitstellt, in abstrakter Weise auf die Hardware zuzugreifen, die andererseits dazu dient, die Ressourcen zu überwachen und zu steuern.

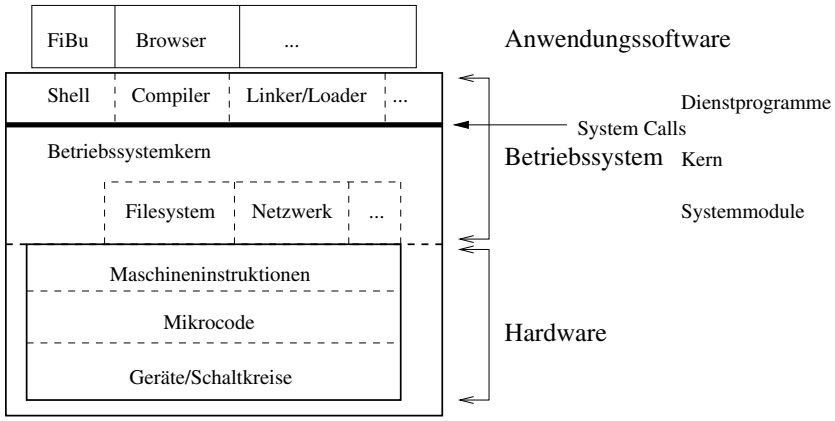

Abb. 1.1. Schichten eines Computersystems

1.1.1 Betriebssystemkern

Das Betriebssystem ist üblicherweise selbst in mehrere Schichten aufgeteilt. In vielen heutigen Betriebssystemen muss zwischen dem Kern und Systemmodulen unterschieden werden. Der Kern enthält die zentralen Aufgaben, die nicht mehr weiter unterteilt werden können. Dazu gehören z.B. die Prozessverwaltung, die Speicherverwaltung oder auch elementare Ein- und Ausgaben.

1.1.2 Systemmodule

Die Aufgabe der Systemmodule liegt darin, komplexe Aufgaben bereitzustellen. Hierzu gehören unter anderem der Zugriff auf Filesysteme[1] oder Netzwerksoftware. Durch die Verlagerung aus dem Kern in diese Module wird ermöglicht, dass ein und dasselbe Betriebssystem auf unterschiedliche Filesysteme zugreifen kann: es kann je nach Bedarf mit unterschiedlichen Modulen installiert werden. So ist es möglich, Linux auf das Filesystem EXT2 oder NTFS oder VFAT zugreifen zu lassen. Darüber hinausgehend können diese Module sogar zeitgleich installiert werden, so dass unterschiedlich formatierte Partitionen gleichzeitig bearbeitet werden können. Der Kern enthält dazu

[1] synonym: „Dateisysteme“

ein virtuelles Filesystem, dessen Zugriffe auf die jeweiligen Module abgebildet werden, die dann ihrerseits auf die konkrete Hardware zugreifen.

Module können bei einigen Betriebssystemen sogar dynamisch im laufenden Betrieb angefordert und geladen sowie wieder entladen werden, so dass der vom Betriebssystem benötigte Platz minimiert wird. Bei Linux muss z.B. zu Beginn nur auf diejenige Partition zugegriffen werden können, auf der das Betriebssystem installiert ist. Diese ist z.B. `ext2`-formatiert. Die Unterstützung für das `ext2`-Filesystem muss dann fest in den Kernel einkompiliert sein, da ansonsten das Betriebssystem die benötigten Informationen mangels Kenntnis des Filesystems nicht lesen kann.[2] Steht auf dem Rechner auch eine Partition für Windows zur Verfügung, die mit NTFS formatiert ist, so wird bei Zugriff auf diese Partition das benötigte Systemmodul automatisch dazugeladen.

1.1.3 Dienstprogramme

Auch dieser Teil der Software ist im weitesten Sinne dem Bereich Betriebssystem zuzuordnen, müssen Shell, Compiler usw. doch die Besonderheiten des jeweiligen Betriebssystems berücksichtigen. So muss ein Compiler „wissen“, auf welcher Adresse das Betriebssystem den Anfang eines Programms erwartet. Entsprechend müssen sich Linker und Loader an die entsprechenden Konventionen halten.

Die Shell kann nur Konstrukte bereitstellen, die das jeweilige Betriebssystem unterstützt. Eine Shell, die Batch-Verarbeitung im Hintergrund anbietet, macht in einem „Betriebssystem“ wie dem alten DOS wenig Sinn.

Was diese Dienstprogramme jedoch von den anderen Teilen des Betriebssystems unterscheidet, sind folgende Eigenschaften:

- Jeder Benutzer kann diese Dienstprogramme aufrufen, sofern er die entsprechenden Rechte besitzt – die (ladbaren) Systemmodule hingegen werden vom Betriebssystemkern selbst aufgerufen oder vom Systemadministrator manuell geladen.
- Sie lassen sich leicht gegen andere Dienstprogramme für das gleiche Betriebssystem austauschen – bei Systemmodulen ist das erheblich schwieriger und kann ausschließlich vom Systemadministrator vorgenommen werden.

1.1.4 Unterschiedliche Arten von Betriebssystemen

Bislang haben wir Betriebssysteme als Erweiterung der Hardware betrachtet. Wenn wir jedoch die Frage nach den einzelnen Aufgaben eines Betriebssystems stellen, kommen wir schnell auf wichtige Unterschiede zu sprechen.

[2] Üblicherweise ist ein Zugriff auf `ext2` zur Zeit Standard und somit fest im Kernel integriert. Jedoch muss bemerkt werden, dass diese Darstellung so zu kurz greift, weil moderne Bootmanager sowie die Inital Ramdisk an dieser Stelle außer Acht gelassen werden.

So lassen sich zunächst folgende große Anforderungsrichtungen unterscheiden:

Batch: Die Aufgaben können routinemäßig ohne Eingriff eines Benutzers durchgeführt werden. Dabei handelt es sich üblicherweise um langwierige Aufgaben, die große Datenmengen bearbeiten müssen oder hohe Rechenzeit erfordern. Hierzu gehören unter anderem statistische Auswertungen, Erstellung von Standard-Berichten, Datenbank-Server usw.

Timesharing: Viele Benutzer können nahezu gleichzeitig auf den Rechner zugreifen und ihre Arbeiten erledigen (Multiuser-Betrieb), so z.B. Edieren und interaktives Testen von Programmen, Datenbankzugriffe, Browser-Benutzung, ...

Transaktions-orientiert: Eine Vielzahl kleiner Aufgaben muss in kürzester Zeit bewältigt werden, so zum Beispiel Banküberweisungen, Buchungssysteme, usw. Dabei muss die Bearbeitung folgende Kriterien erfüllen: Atomarität, Konsistenz, Isolation und Dauerhaftigkeit (englisch ACID), d.h. die Bearbeitung muss entweder vollständig ablaufen oder keinerlei Änderung hinterlassen.

Echtzeit: Bei der Überwachung von chemischen oder technischen Prozessen muss das System in einer durch das jeweilige Problem vorgegebenen Zeit auf das Auftreten von Ereignissen reagieren. Dabei kann zwischen harten und weichen Echtzeitsystemen unterschieden werden:

- Harte Echtzeitsysteme:
  Die Reaktionszeit darf bei manchen Prozessen auf keinen Fall überschritten werden, weil sonst ernsthafte Schäden auftreten können (harte Echtzeitsysteme). Reagiert ein Autopilot in einem Flugzeug zu spät auf das Erreichen einer kritischen Fluglage, kann es zum Absturz kommen.
- Weiche Echtzeitsysteme:
  Bei anderen Systemen, so bei Audio oder Multimedia-Systemen, sind gewisse Toleranzen hinsichtlich der Abweichung von der Reaktionszeit erlaubt. Diese Systeme nennt man weiche Echtzeitsysteme.

Neben diesen großen Richtungen gibt es auch Hardware-Anforderungen, die Betriebssysteme unterscheiden lassen:

- Hohe Ein- und Ausgabe-Bandbreite und Verwaltung riesiger Datenmengen: diese Anforderungen findet man üblicherweise bei größeren Rechensystemen[3]. Große Server müssen häufig diese Anforderung erfüllen.
- Server: Server laufen üblicherweise auf großen Rechnersystemen, Workstations oder großen PCs. Typisches Kennzeichen ist, dass viele Benutzer über ein Netz gleichzeitig auf definierte Dienste zugreifen können, z.B. Druckdienste, Datenbankzugriff, Fileserver usw.

[3] Früher sprach man von Mainframes oder „Großrechnern“, doch sind diese Begriffe heute eher negativ belegt, obwohl heutige große Server in diese Kategorie fallen.

- Workstations: Die wichtige Anforderung besteht darin, interaktive Schnittstellen für einen einzelnen Benutzer zur Verfügung zu stellen. Tastatur, Maus, grafische Oberfläche und Lautsprecher müssen zumindest unterstützt werden. Heute zählt in der Regel auch eine entsprechende Netzanbindung zu den Aufgaben.
- Spezielle Hardware: Hierzu zählen Multiprozessor-Betriebssysteme, aber auch embedded („verkleinerte“) Betriebssysteme für PDAs (Personal Digital Assistent) oder Spezialhardware.

1.2 Computer-Hardware

Wenn Betriebssysteme in gewisser Weise als „Erweiterung“ der Hardware angesehen werden können, dann sind Betriebssystem und Hardware sehr eng verzahnt. Abbildung 1.1 auf S. 2 zeigt, dass das Betriebssystem auf den Maschineninstruktionen aufsetzt. Am anderen Ende der Hardware stehen Schaltkreise bzw. physische Geräte. Aus diesem Grunde wollen wir uns einen Überblick über den Aufbau eines Rechners verschaffen. Als Beispiel soll die Architektur eines PCs herangezogen werden. Wir wissen, dass der PC über eine CPU, über Speicher, eine Grafikkarte usw. verfügt. Doch wie arbeiten diese Bestandteile zusammen? Was lässt sich über ihren Aufbau, ihre Funktion sagen?

Abbildung 1.2 skizziert den Aufbau eines heutigen PCs. Die gestrichelten Komponenten sind nicht notwendigerweise vorhanden. Die Linien stellen Busse dar, die die einzelnen Komponenten miteinander verbinden. Die Strichstärke der Busse soll die unterschiedlichen Geschwindigkeiten verdeutlichen, mit der Daten über den jeweiligen Bus ausgetauscht werden können.

Die Architektur eines größeren Rechensystems ist in der Regel wesentlich komplexer, auch wenn Komponenten wie CPU, Cache, Speicher, Busse, Controller usw. dort ebenfalls zu finden sind.

1.2.1 CPU

Ein Programm besteht aus einer Folge von Maschineninstruktionen, die in aufeinanderfolgenden Speicheradressen abgelegt sind. Aufgabe der CPU ist es, die Programme Schritt für Schritt umzusetzen.

Bei unserer folgenden, stark vereinfachten Betrachtung legen wir die x86-Architektur zu Grunde. Die CPU ist in Rechenwerke und verschiedene Register unterteilt. Eines dieser Register ist der Befehlszähler (Program Counter), der auf die Adresse[4] der gerade auszuführenden Instruktion zeigt. Daraufhin liest die CPU diese Maschineninstruktion aus der entsprechenden Stelle im Speicher und führt sie aus. Handelt es sich nicht um eine Operation, die eine

[4] Der Speicher kann Byte-weise adressiert werden.

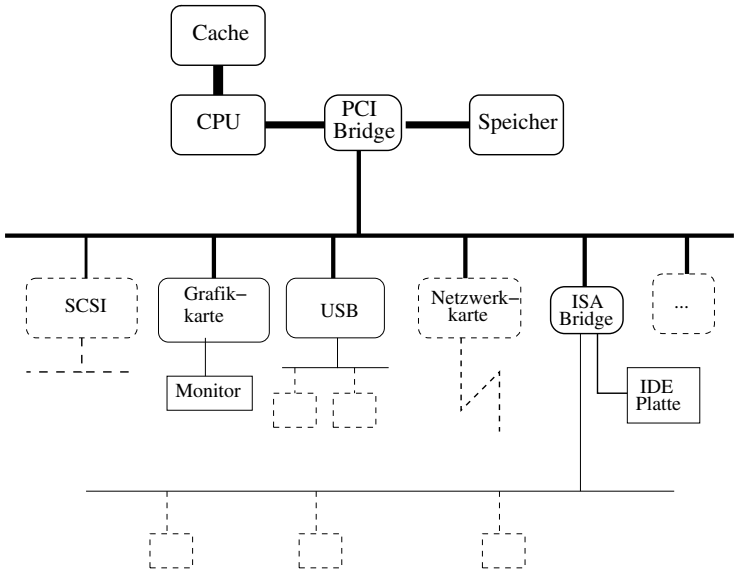

Abb. 1.2. Architektur eines PCs
Die Darstellung ist stark vereinfacht: die „PCI-Bridge“ besteht heute aus zwei Teilen, der North-Bridge, die für die Speicherzugriffe und die Grafikkarte zuständig ist, und die damit verbundene South-Bridge, die die langsameren Verbindungen wie PCI, USB, UDMA/ATA usw. bedient.
Die Strichstärke der Busse versinnbildlicht die Übertragungsgeschwindigkeit.

Verzweigung des Kontrollflusses[5] bewirkt, so wird nach der Ausführung der Befehlszähler erhöht. Da Instruktionen ein, zwei oder sogar mehrere Byte lang sein können, wird der Befehlszähler genau um die Anzahl der Byte erhöht, die die gerade ausgeführte Instruktion einnimmt. Anschließend wird die nächste Instruktion eingelesen. Handelt es sich hingegen um eine Verzweigung des Kontrollflusses, so wird die ermittelte Adresse der danach auszuführenden Instruktion in den Befehlszähler geladen.

Allgemeine Register dienen zur Aufnahme von Daten und Adressen. Diese Register können direkt adressiert und manipuliert werden.

Weitere wichtige Register sind

- Stackregister: Dieses dient zum Adressieren eines Stacks. Mit Hilfe eines Stacks lässt sich z.B. die Wertübergabe bei Prozeduraufrufen implementieren.
- Coderegister: Hiermit wird das Segment adressiert, das den derzeit ausgeführten Maschinencode enthält.
- Datensegmentregister: Jedes dieser Register dient dazu, ein Datensegment zu adressieren.

[5] Z.B. Sprungbefehl oder Prozeduraufruf.

- GDTR, LDTR, IDTR: Register, die beim Aufbau von Pagetables und Interrupt-Deskriptor-Tabellen benötigt werden.
- Statusregister: Vergleiche führen dazu, dass einzelne Bits dieses Registers gesetzt werden. Diese Werte werden benutzt, um bei bestimmten Maschineninstruktionen eine Verzweigung auszulösen.

Moderne CPUs sind komplexer aufgebaut als diese kurze Beschreibung vermuten lässt; in dem betrachteten Kontext wird aber keine detailliertere Betrachtung benötigt.

Das Betriebssystem muss an die jeweilige CPU angepasst sein. Welche Register tatsächlich zur Verfügung stehen und wie sie eingesetzt werden, ist von der CPU abhängig. Wir müssen also erwarten, dass zumindest ein Teil der Betriebssystem-Software sehr CPU-nah in Assembler programmiert ist.

Mit diesen Überlegungen können wir verstehen, was passiert, wenn eine einzige Anwendung zu einem Zeitpunkt auf einem Rechner läuft. Aber in unserer täglichen Arbeit am PC haben wir in der Regel mehrere Anwendungen gleichzeitig laufen. Wie kann die CPU diese Anwendungen „gleichzeitig“ bedienen? Wie kann der Begriff „Anwendung“ aus Sicht eines Betriebssystems beschrieben werden? Was geschieht, wenn eine neue Anwendung gestartet wird, wenn eine laufende Anwendung beendet wird. Diese – und weitere – Probleme werden im Kap. 3 angesprochen.

1.2.2 Busse

Tatsächlich kommuniziert die CPU nicht mit dem Speicher oder der Grafikkarte direkt, sondern über die PCI-Bridge bzw. über die North- und South-Bridge[6], die den entsprechenden Datenstrom je nach Anweisung entweder auf den Speicher oder auf den PCI-, USB-Bus usw. abbilden. Auf der anderen Seite ist die CPU über einen sehr schnellen Bus mit dem internen Cache verbunden.

Die PCI-Bridge ist mit einem Bus mit der CPU verbunden, mit einem anderen Bus mit dem Speicher. Als dritter Bus verlässt der PCI-Bus diesen Baustein. Diese Lösung ermöglicht es, dass der Speicher auch direkt mit den externen Controllern Daten austauschen kann, ohne dass die CPU dabei aktiv beteiligt sein muss.

In den PCI-Bus sind diverse Controller eingesteckt, so die Grafik-Karte, in der Regel eine Netzwerkkarte und ein USB-Controller, an dem unter anderem Maus, Tastatur, Laufwerk und Scanner angeschlossen werden können. Neben weiteren dort ggf. eingehängten Bussen wie SCSI und Firewire gibt es noch die ISA-Bridge. Diese dient dazu, den Übergang in den ältesten PC-Bus vorzunehmen, der aus Gründen der Kompatibilität zu alter Hardware noch vorhanden ist. Hier werden parallele Drucker angeschlossen.

[6] Die derzeitige Variante besteht im Einsatz von North- und South-Bridge; zur Vereinfachung der Darstellung wird die ältere Version der PCI-Bridge herangezogen.

Ebenfalls an der ISA-Bridge angeschlossen ist der IDE-Bus, der Platten und CD- bzw. DVD-Laufwerke unterstützt.

Das Betriebssystem muss die Busse kennen und ansprechen können, damit sich die angeschlossenen Geräte beim Start entsprechend konfigurieren lassen.

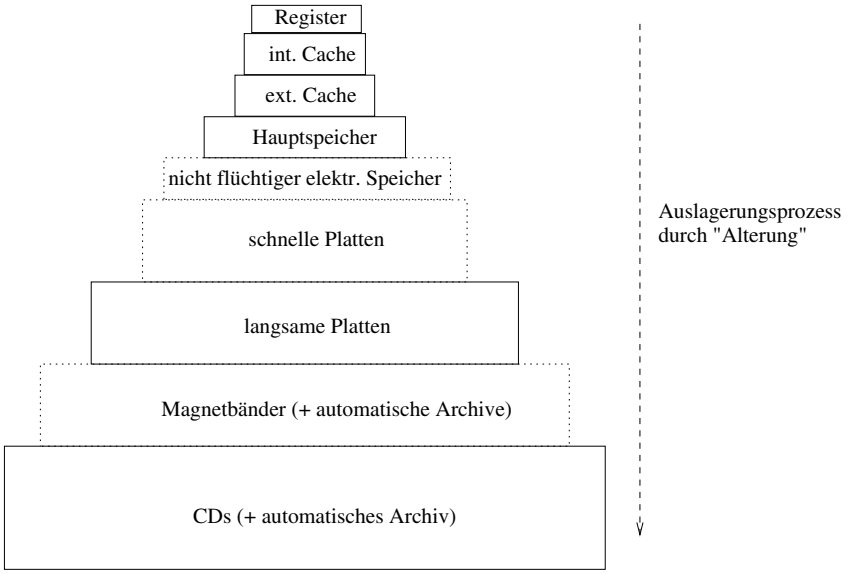

Abb. 1.3. Speicherhierarchie
Gepunktet sind diejenigen Speicherschichten, die beim PC in der Regel nicht vorhanden sind.

1.2.3 Speicher

Der Speicher nimmt Daten und auszuführende Programme auf. Zum Speicher zählt nicht nur der Hauptspeicher des Rechners, tatsächlich bildet der Speicher eine durch Busse verbundene Hierarchie (vgl. Abb. 1.3). Diese beginnt bei den ganz schnellen CPU-Registern, auf die die CPU ohne Verzug zugreifen kann, führt über den internen und externen Cache, den eigentlichen Hauptspeicher, nicht flüchtigen elektronischen Speicher[7], schnelle und langsame Magnetplatten, Magnetbänder bis hin zu CDs und DVDs. Der Grund liegt im Preis/Leistungsverhältnis: je schneller der Speicher, desto teurer ist er.

Es gibt Betriebssysteme im Bereich großer Rechenanlagen, die automatisch und für den Benutzer transparent diese Speicherhierarchie ausnutzen,

[7] Nur in großen Rechenanlagen vorhanden

um Daten, die längere Zeit nicht benutzt wurden, in die langsameren Schichten „altern“ zu lassen. Erfolgt jedoch ein Zugriff auf diese Daten, so werden sie – unabhängig von der Schicht, in der sie sich gerade befinden – so schnell wie möglich in den Vordergrund, d.h. in den Hauptspeicher geholt.

Bei der täglichen Benutzung unseres PCs lassen wir mehrere Programme gleichzeitig laufen – z.B. einen Kalender, einen Editor, einen Compiler usw. Der ausführbare Code und die Daten werden im Speicher gehalten. Damit ergeben sich sofort eine Reihe von Fragen:

- Wie verwaltet das Betriebssystem den Hauptspeicher?
- Wie schützt das Betriebssystem die im Hauptspeicher befindlichen Teile einer Anwendung gegen „Übergriffe“ durch andere Anwendungen?

Auf diese Fragen wird im Kap. 5 eingegangen.

Betrachten wir nicht nur den Hauptspeicher, sondern die Speicherhierarchie, dann stellen sich entsprechende Fragen für die Verwaltung der Platten. Genauere Antworten hierauf gibt das Kap. 8.

1.2.4 Ein- und Ausgabegeräte

Abbildung 1.2 und die zugehörige Diskussion im Abschn. 1.2.2 deutet eine Reihe von Ein- und Ausgabegeräten an. So finden wir z.B. die „Grafik-Karte“ und als zugehöriges Gerät den Monitor, oder den USB-Controller und angedeutet die Maus bzw. Tastatur. Alle diese Karten und Kontroller sind mit spezialisierten kleinen „Rechnern“ ausgestattet.

Dieser zweigeteilte Aufbau ist typisch: damit die CPU nicht die gesamte Arbeit selbst erledigen muss, werden intelligente Chips oder Platinen – oder sogar ganze Vorrechner – bereitgestellt, die einen eigenen Befehlssatz besitzen.

Wie das Betriebssystem die Hardware vor den Anwendungsprogrammen verbirgt, so verdecken die Controller die Geräteschnittstellen vor dem Betriebssystem. Damit andererseits das Betriebssystem nicht auf die Vielzahl unterschiedlicher Controller und deren unterschiedliche Befehlssätze angepasst werden muss, werden zu jedem Controller Treiber für die unterstützten Betriebssysteme geliefert. Diese Treiber müssen in das Betriebssystem integriert werden. Je nach Betriebssystem können Treiber statisch bei der Kompilation des Kernels eingebunden oder dynamisch während der Laufzeit ge- oder entladen werden. Dynamisches Laden und Entladen ist für Hotplug erforderlich, d.h. für Geräte, die während der Laufzeit angeschlossen bzw. entfernt werden dürfen.

Betrachten wir folgende Situation: eine Anwendung möchte einen Datensatz von einer Platte lesen. Dazu schickt die CPU dem IDE-Controller unter Verwendung des Treibers einen I/O-Befehl[8], und der Controller leistet dann

[8] Genauer: der Befehl wird in ein Register des Controllers geschrieben. Diese Register können entweder in den Speicher eingeblendet sein, d.h. Kommunikation mit dem Controller wird zu einem normalen Speicherzugriff, oder durch spezielle I/O-Befehle (Input/Output = Ein-/Ausgabe) adressiert werden.

die in der Regel komplexe zeitaufwändige Arbeit, ohne dass die CPU weiter eingreifen muss. In diesem Falle müssen zunächst der Zylinder und Block auf der Platte bestimmt werden, in dem der Datensatz steht, der Plattenarm muss positioniert und dann gewartet werden, bis der Block bei der Rotation unter dem Lesemechanismus des Plattenarms erscheint. Danach kann gelesen werden. Jetzt hat der Controller die Daten – und nun?

Wie kommt die Anwendung an die Daten? Drei wesentliche Techniken werden eingesetzt:

- „Busy waiting“: Der Treiber wartet in einer Endlosschleife darauf, dass der Controller in einem Register anzeigt, dass die Daten zum Auslesen bereit stehen. Danach werden die Daten ausgelesen, an die gewünschte Stelle im Speicher geschrieben und die Anwendung erhält wieder die Kontrolle und kann mit ihrer Arbeit fortfahren.
- „Interrupt-gesteuert“: Der Treiber initialisiert die Aufgabe und wartet dann auf einen Interrupt (Unterbrechung) durch den Controller. Anstatt nun – wie im vorigen Vorgehen – die CPU zu beschäftigen, wartet der Treiber darauf, durch den Interrupt „geweckt“ zu werden. Das Betriebssystem bekommt die Kontrolle und kann bis zum Eintreffen des Interrupts andere Anwendungen bedienen.
  Wenn der Controller mit seiner Arbeit fertig ist und den erwarteten Interrupt erzeugt, wird die CPU in ihrer momentanen Arbeit unterbrochen, ihr Zustand wird gerettet[9] und die Unterbrechungsroutine wird aufgerufen. Diese holt in unserem Falle die Daten vom Controller und legt sie an entsprechender Stelle im Speicher ab. Anschließend wird der alte Zustand der CPU wieder hergestellt und die unterbrochene Arbeit fortgesetzt.
- „DMA“ (Direct Memory Access): dazu wird ein spezieller Baustein benötigt, der Daten direkt, ohne Verwendung der CPU, zwischen Speicher und Controller transportieren kann. Dieser Baustein muss mit den entsprechenden Informationen initialisiert werden: Speicherstelle, Anzahl der Bytes, Controller, Richtung. Nach Beendigung erzeugt der DMA-Baustein einen Interrupt.

Die erste Methode („busy waiting“) ist am einfachsten zu verwirklichen, belastet aber den Prozessor. Gleichzeitige Bearbeitung von verschiedenen Anwendungen wird dadurch stark behindert.

Die zweite Methode setzt zusätzliche Hardware voraus: einen Interrupt-Controller sowie eigene Leitungen für die Versendung des Interrupts. Dafür wird die CPU frei, während der Wartezeit der einen Anwendung andere Anwendungen zu bedienen.

Die dritte Methode erfordert noch mehr Hardware-Aufwand: zusätzlich zu einem Interrupt-Controller wird ein DMA-Baustein benötigt. Der Vorteil liegt darin, dass die CPU nun auch nicht mehr benötigt wird, um Daten zwischen Speicher und Controller zu transportieren.

[9] Das dient dazu, dass anschließend die unterbrochene Arbeit fortgesetzt werden kann.

Über Ein-/Ausgabe bzgl. Dateien wird in Kap. 8 weiter nachgedacht, System Calls und Interrupts werden im Abschn. 1.3 und in den Kapiteln 2 und 7 näher betrachtet.

1.3 System Calls und Interrupts

Das Betriebssystem stellt Dienste (Funktionen) bereit, die es den Anwendungen ermöglichen, die gewünschte Funktionalität zu bekommen. Es stellt sich die Frage, wie eine Anwendung darauf zugreifen kann. Zu Beginn der ersten Rechner mit „Betriebssystemen“ – und die Geschichte wiederholte sich mit dem Aufkommen der PCs im DOS – waren im Speicher feste Einsprungadressen für die Funktionen des Betriebssystems vorgesehen. Die Anwendung führte eine `call`-Anweisung auf die jeweilige Adresse aus, nachdem die Register mit den benötigten Parametern gefüllt waren. Am Ende der jeweiligen Betriebssystemroutine wurde eine `ret`-Anweisung (Return) ausgeführt, die Anwendung konnte mit ihrer Arbeit weitermachen.

Der Nachteil dieses Vorgehens liegt auf der Hand: ohne Speicherschutz konnten die Funktionen des Betriebssystems von fehlerhaften (oder auch böswilligen) Anwendungen überschrieben werden und anschließend eine andere Funktionalität aufweisen.

Für eine sichere Benutzung wurde deshalb schon früh in der Entwicklung der Betriebssysteme ein Konzept entwickelt, das Anwendungen den Zugriff auf die Funktionalität erlaubt, obwohl die Kontrolle ausschließlich beim Betriebssystem liegt. Dies setzt eine geeignete Hardware-Erweiterung voraus: der Speicherbereich, in den das Betriebssystem geladen wird, muss gegen Zugriffe von Anwendungen geschützt werden und der Prozessor muss zumindest zwei Privilegien unterscheiden können: den User Mode (normaler Modus), in dem die Anwendungen laufen, und den Kernel Mode (privilegierter Modus), in dem die Betriebssystemroutinen ablaufen. Dabei kann insbesondere der Zugriff auf Geräte, d.h. I/O-Aufrufe, nur im privilegierten Modus erfolgen.

1.3.1 Der eigentliche Aufruf

Anwendungen laufen immer im User Mode und müssen sich deshalb der System Calls bedienen, um auf die angebotene Funktionalität des Betriebssystems zuzugreifen. Das prinzipielle Vorgehen beim System Call bei den verschiedenen Betriebssystemen ist ähnlich:

- Die Argumente werden in Registern oder in einem Speicherblock in vorgeschriebener Weise abgelegt.
- Sodann wird, je nach Architektur der Hardware, ein spezieller Systemaufruf oder ein Interrupt erzeugt. Genau dabei wird der Prozessor in den Kernel Mode versetzt und das Betriebssystem erhält die Kontrolle.
- Das Betriebssystem prüft dann, welcher System Call aufgerufen werden soll, und ruft diesen auf.

- Der System Call kann eine Reihe von Überprüfungen vornehmen, bevor er die eigentliche Arbeit leistet: Soll eine Datei geöffnet werden, so kann der System Call zunächst prüfen, ob der Benutzer auch dazu berechtigt ist. Auf diese Weise kann das Betriebssystem Benutzerrechte überwachen.
- Nach Abarbeitung des System Calls kann das Betriebssystem einer anderen Anwendung die Kontrolle übergeben – dies wird in der Regel bei heute üblichen Betriebssystemen geschehen, wenn erst noch auf Daten von einer Ein-/Ausgabe gewartet werden muss – oder zu der Anwendung zurückkehren, die den System Call ausgelöst hat. In jedem Falle schaltet der Prozessor mit der Übergabe der Kontrolle an eine Anwendung in den User Mode zurück.

1.3.2 Interrupts

Wir haben Interrupts bereits auf S. 10 kennengelernt. Sie spielen eine große Rolle bei der Bearbeitung. So dienen sie nicht nur zum Signalisieren des Endes einer I/O-Anforderung, sie werden auch eingesetzt, um Fehler abzufangen und darauf zu reagieren: „Division durch Null“ ist nur ein Beispiel von vielen.

Aber auch andere wichtige Aufgaben werden damit gelöst. Zum Beispiel benutzen viele heutige Betriebssysteme den Speicher so, dass nicht die gesamte Anwendung auf einmal in den Hauptspeicher geladen werden muss, sondern nur ein Teil. Erzeugt die CPU dann eine Adresse, auf die die Anwendung zugreifen will, so prüft die Hardware

1. ob es sich um eine Adresse handelt, die zum Adressraum der Anwendung gehört. Ist diese nicht der Fall, so wird ein `Illegal Address Interrupt` erzeugt. Das Betriebssystem beendet daraufhin die Anwendung mit einer Fehlermeldung,
2. ob diese Adresse bereits im Hauptspeicher geladen ist. Wenn nicht, wird ein `Pagefault Interrupt` erzeugt, der dazu führt, dass das Betriebssystem den entsprechenden Teil der Anwendung an geeigneter Stelle in den Hauptspeicher lädt.

Nachdem nun die Adresse im Hauptspeicher abgebildet ist, kann darauf zugegriffen werden.

Wir haben gesehen, wie Anwendungen bearbeitet werden können, deren Speicherbedarf größer als der Hauptspeicher ist. Näheres dazu in Kap. 5.

Ein weiterer Einsatz von Interrupts ist für viele heutige Betriebssysteme wichtig. Denken wir für einen kurzen Moment daran, dass wir in der Regel mehrere Anwendungen zugleich auf unserem Rechner laufen lassen. Wie kann das sicher funktionieren?

Auf damaligen Großrechnern war sicheres Multitasking („gleichzeitige“ Bearbeitung vieler Anwendungen) längst Standard, als mit Windows 3.1 auf dem

PC diese Möglichkeit eröffnet wurde.[10] Die Lösung in Windows 3.1 lautete: durch Aufruf von System Calls gab eine Anwendung freiwillig die Kontrolle ab und ermöglichte es anderen Anwendungen, die CPU zu bekommen. Was passiert aber, wenn ein Programm aus Versehen oder absichtlich eine Endlosschleife enthält, die keine System Calls beinhaltet? Deshalb schlug man auch bei den PC-Betriebssystemen den Weg ein, der viele Jahre zuvor schon bei Großrechnern beschritten worden war: die Hardware-Uhr erzeugt in regelmäßigen kurzen Abständen einen Timer-Interrupt, dadurch erhält das Betriebssystem die Kontrolle und kann einer Anwendung die CPU entziehen, um sie einer anderen zuzuteilen. Dabei entsteht nun folgendes Problem: wie kann eine Anwendung, nachdem ihr die CPU entzogen wurde, richtig weiterarbeiten, wenn sie die CPU wieder zugeteilt bekommt. Um das zu verstehen, müssen wir genauer über den Begriff „Anwendungen“ nachdenken. Offensichtlich gehört mehr dazu, als nur den Programmcode in den Speicher zu laden und die CPU diesen dann abarbeiten zu lassen. Solange die CPU für die Anwendung tätig ist, mag diese Sicht einigermaßen ausreichen, doch was ist, wenn sie entzogen wird? Damit die Anwendung an genau der Stelle weitermachen kann, an der die CPU entzogen wurde, müssen Informationen aufbewahrt werden: Der Zustand der CPU-Register muss gerettet werden und bei erneuter Zuteilung müssen die Register wieder mit der geretteten Information initialisiert werden. Programmcode, Speicherplatz für die CPU-Register, Datensegmente, Prioritäten, Ressourcen (vgl. Abschn. 1.4) und Rechte fasst man im Begriff „Prozess“ zusammen. Der Algorithmus, der vom Betriebssystem verwendet wird, um nach einem derartigen Interrupt denjenigen Prozess auszuwählen, der die CPU bekommen soll, wird als Scheduling bezeichnet. Kap. 3 betrachtet Prozesse detaillierter, Kapitel 4 geht auf das Scheduling ein.

1.4 Ressourcen

Ein Prozess benötigt in der Regel mehr als nur Hauptspeicher und – wenn er aktiv ist – die CPU. Tatsächlich greift er auch auf Geräte, Dateien usw. zu.

All diese Objekte, die ein Rechner zur Verfügung stellt, werden mit dem Begriff Ressource belegt. So kann es Ressourcen geben, die mehrfach im System vorhanden sind – beispielsweise mehrere Plattenpartitionen, zwei DVD-Laufwerke – und Ressourcen, die aus wirtschaftlichen oder Nutzer-bedingten Ansprüchen nur einmal zur Verfügung stehen – CPU (es sei denn, wir haben ein Mehrprozessor-System), Plotter[11] usw.

Neben der Anzahl vorhandener Exemplare einer Ressource ist von großer Bedeutung, ob man die Ressource einem Prozess entziehen kann. Wie wir im

[10] Es sei angemerkt, dass das Betriebssystem OS/2 bereits „echtes“ Multitasking auf dem PC eingeführt hatte, jedoch wurde dieses Betriebssystem bei Weitem nicht so bekannt wie Windows.

[11] Natürlich kann es in einem Rechner mehrere geben. Aber in der Regel wird man in einem PC nur einen vorfinden.

vorigen Abschnitt gesehen haben, kann das Betriebssystem einem Prozess die CPU entziehen, um sie einem anderen Prozess zuzuteilen. Doch gilt das nicht für alle Ressourcen: Wenn ein Prozess gerade Daten auf den DVD-Writer sichert und das Betriebssystem in dieser Zeit dem Prozess den DVD-Writer entzieht, dann wäre die DVD unwiederbringlich zerstört.

Besondere Bedeutung bekommt der Begriff Ressource im Abschn. 6.2.

1.5 Zusammenfassung

Wichtige Konzepte aus dem Gebiet „Betriebssysteme“ sind angerissen worden:

- Die enge Verknüpfung von Hardware und Betriebssystem,
- System Calls und Interrupts, um Dienste des Betriebssystems anzufordern,
- der Begriff „Prozess“ als Zusammenfassung aller momentan benötigten Ressourcen, Rechte, Prioritäten usw., um eine Anwendung zu bearbeiten,
- Timer-Interrupt und Scheduling als Methode des Betriebssystems, Prozesse „gleichzeitig“ zu bearbeiten,
- prinzipielle Vorgehensweisen beim I/O,
- Hauptspeicherverwaltung, um die Grenzen des physischen Hauptspeichers zu überwinden,
- das Filesystem sowie die
- Modularisierung des Systems.

Die folgenden Kapitel sollen dazu dienen, verschiedene eben betrachtete Aspekte zu vertiefen, indem sie jeweils einen besonderen Schwerpunkt herausgreifen. In jedem Kapitel werden zunächst allgemeine Vorgehensweisen vorgestellt, die man als grundlegende „Prinzipien“ für die Implementierung des jeweiligen Aspekts bezeichnen könnte. Doch konkrete Implementierungen weichen üblicherweise ein wenig von diesen Prinzipien ab. Die sich anschließenden Abschnitte zeigen unterschiedlich stark detailliert, wie die konkrete Implementierung in Linux aussieht.

Ein besonderes Problem für Linux stellt sich dadurch, dass Linux nicht auf eine Hardware-Plattform ausgerichtet ist, sondern auf einer großen Zahl unterschiedlicher Plattformen performant laufen soll. Hier erweist sich die eben erwähnte „enge Verknüpfung zwischen Hardware und Betriebssystem“ als besondere Schwierigkeit. Linux packt dieses Problem so an, dass der Quellcode in einen allgemeinen Hardware-unabhängigen Teil und in Hardware-spezifische Teile gegliedert wird. In diesem Buch werden wir uns auf die x86-Architektur beschränken, da diese üblicherweise zur Verfügung stehen wird.

Um hinreichend performant zu sein, greift Linux in den Hardware-spezifischen Code-Teilen zum Trick, besonders zeitkritische oder besonders Hardware-abhängige Funktionen in Assembler zu programmieren. Dies wird durch den verwendeten C-Compiler gut unterstützt. Die im folgenden gewählte Betrachtungsweise wird üblicherweise auf der Ebene des C-Codes bleiben.