Rolf Klein, Christian Icking, Lihong Ma

Betriebssysteme und Rechnernetze

Kurseinheit 1:
Geräte und Prozesse

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Der Inhalt dieses Studienbriefs wird gedruckt auf Recyclingpapier (80 g/m^2, weiß), hergestellt aus 100 % Altpapier.

Inhalt

Teil I: Betriebssysteme

Teil II: Rechnernetze

Die Autoren

Univ.-Prof. Dr. rer. nat. Rolf Klein

Studium der Mathematik und mathematischen Logik an der Universität Münster zum Dipl.-Math. 1978, dann Universität Erlangen-Nürnberg, hier Promotion in Mathematik 1982, danach Universität Karlsruhe, University of Waterloo in Kanada, Universität Freiburg, dort Habilitation in Informatik 1989. Hochschulprofessor an der Universität Essen, der FernUniversität in Hagen, seit 2000 an der Universität Bonn, seit 2019 emeritiert.

Univ.-Prof. Dr.-Ing. Jörg Haake

Studium der Informatik an der Universität Dortmund zum Dipl.-Inform. 1987, dann BCT GmbH und Institut für Integrierte Publikations- und Informationssysteme der GMD in Darmstadt, Promotion in Informatik 1995 an der Technischen Universität Darmstadt, Bereichsleiter am Fraunhofer Institut für Integrierte Publikations- und Informationssysteme (IPSI) in Darmstadt, seit 2001 Professor für Praktische Informatik an der FernUniversität in Hagen.

apl. Prof. Dr. rer. nat. Christian Icking

Studium der Mathematik und Informatik an der Universität Münster und der Université Pierre und Marie Curie Paris VI zur Maîtrise in Mathematik 1984, dann auch École Nationale Supérieure de Techniques Avançées Paris zum DEA (Diplom) Informatik 1985, danach Universität Karlsruhe, Universität Freiburg, Universität Essen und FernUniversität in Hagen, hier Promotion und Habilitation in Informatik 1994 und 2002, außerplanmäßiger Professor 2010.

Dr. rer. nat. Lihong Ma

Studium der Mathematik an der Universität Yunnan, Bachelor in Mathematik 1984, dann Technische Universität Kunming, Universität Karlsruhe, Universität Freiburg, dort Dipl.-Math. 1990, danach Universität Essen und FernUniversität in Hagen, hier Promotion in Informatik 2000, wissenschaftliche Mitarbeiterin.

Kurseinheit 1

Geräte und Prozesse

1.1 Einführung

Diese Einführung soll Ihnen einen Überblick darüber geben, worum es im vorliegenden Kurs geht und an welchem Platz die Kursinhalte in der Informatik als Wissenschaft angesiedelt sind.

Drei Ursachen für Computerverbreitung:

Seit den 1980er Jahren hat der Computer nicht nur die Berufswelt von Grund auf verändert, er hat auch in den Bereichen Bildung und Freizeit Einzug gehalten. Diese Entwicklung wurde durch verschiedene Faktoren ermöglicht und begünstigt:

1. Hardware preiswert und schnell

Zum einen ist die *Hardware* der Computer – dazu gehören unter anderem der Prozessor, die Hauptspeicherbausteine und die Festspeicher – immer leistungsfähiger und preiswerter geworden. Erst dadurch wurden Computer für kleine Betriebe und für private Benutzer überhaupt erschwinglich.

Zum anderen ist der Umgang mit dem Computer immer einfacher geworden: Wer gelegentlich ein Textverarbeitungssystem oder ein Spielprogramm benutzen möchte, kommt mit den komplexen technischen Details der Hardware überhaupt nicht in Berührung und braucht sie deshalb nicht zu kennen.

Hier hat sich eine ähnliche Entwicklung vollzogen wie bei den Kraftfahrzeugen: Um einen Oldtimer zu starten, musste man Zündzeitpunkt und Gemischanreicherung sorgfältig einstellen und die technischen Zusammenhänge kennen – heute wird dem Fahrer diese Aufgabe von elektronischen Motormanagementsystemen abgenommen. Beim modernen Computer übernimmt das *Betriebssystem* diese Funktion. Es bildet eine Schicht zwischen der Rechnerhardware und den Anwendungsprogrammen. Das Betriebssystem trennt die Anwendungen von der Hardware, ermöglicht es ihnen aber auch, in kontrollierter Weise mit der Hardware zu interagieren.

2. Betriebssystem verwaltet Hardware

Dieses Konzept der Unabhängigkeit zwischen Software und Hardware bildete eine wesentliche Voraussetzung für die Entstehung des *Personalcomputers* (PC)[1], der von Jedermann bedient werden kann, und für die Entwicklung kostengünstiger Anwendungsprogramme.

[1] Darunter verstehen wir einen Rechner, der einem einzelnen Benutzer zum persönlichen Gebrauch zur Verfügung steht, typische Betriebssysteme sind Windows, macOS und Linux.

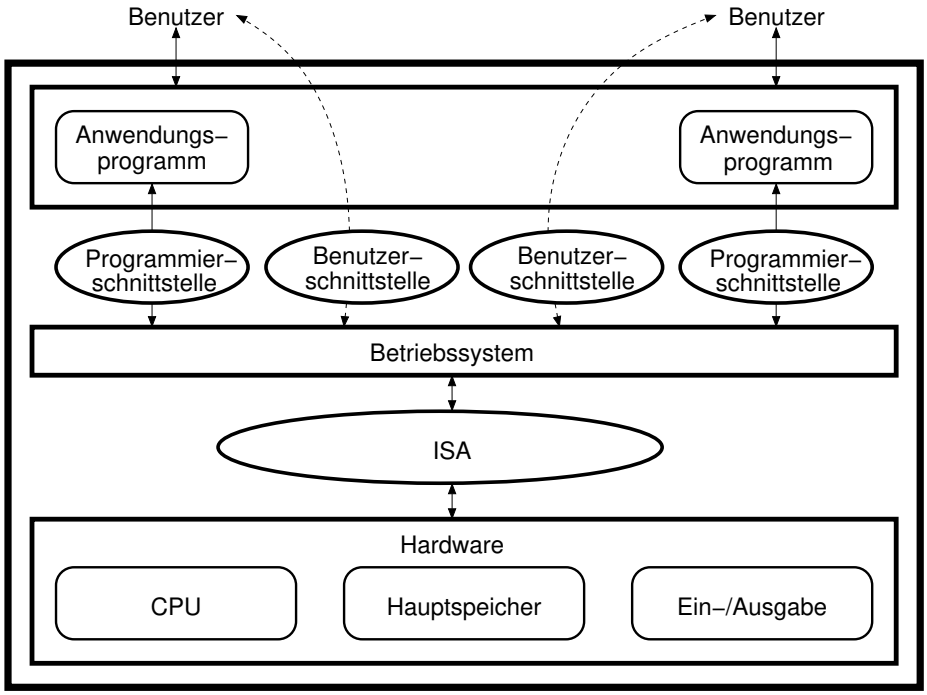

Abbildung 1.1: Einfaches Schichtenmodell eines Computersystems: Das Betriebssystem als Vermittler zwischen der Rechnerhardware (unten) und der Anwendungssoftware und den Benutzern (oben).

Schicht

Ein *Computersystem* besteht aus mehreren *Schichten*. Ein vereinfachtes Schichtenmodell ist in Abbildung 1.1 dargestellt:

- Die oberste Schicht ist eine Menge von *Anwendungsprogrammen*[2] wie z. B. E-Mail-Programme und Web-Browser, die von einem normalen Benutzer gestartet und ausgeführt werden können. Ein Benutzer sieht einen Computer als eine Menge von Anwendungsprogrammen.
- Die Schicht *Betriebssystem* ist eine Menge von besonderen Programmen, die für Benutzer nicht direkt sichtbar sind und deren Aufgabe es ist, die Hardware wie z. B. CPU, Speicher und Ein-/Ausgabegeräte so zu steuern, dass die Ausführung von Anwendungsprogrammen möglich wird.
- Die unterste Schicht ist die Hardware, die aus Prozessoren, Speichern und Ein-/Ausgabegeräten besteht.

Schnittstelle

Zwischen zwei Schichten ist in Abbildung 1.1 jeweils eine sogenannte *Schnittstelle* dargestellt, dies ist eine Menge von Operationen. Jede Schicht bietet der direkt darüber liegenden Schicht eine Menge von *Services* (Diensten) an, ein Service besteht aus einer Menge von Programmen. Eine Schicht kann einen Service der direkt darunter liegenden Schicht nur über diese Schnittstelle erhalten. D. h. eine Schnittstelle definiert, wie eine Schicht auf Dienste der direkt darunter liegenden Schicht zugreifen kann, sie ermöglicht die Kommunikation zwischen zwei benachbarten Schichten.

[2] Anwendungsprogramme werden manchmal auch *Anwender*programme, *Applikationen* oder *Apps* genannt.

Zwei *Schnittstellen* werden hierfür vom Betriebssystem zur Verfügung gestellt:

- eine *Programmierschnittstelle* für die Kommunikation der Programme mit dem Betriebssystem und
- eine *Benutzerschnittstelle*[3] für die Kommunikation der Benutzer mit dem Betriebssystem in Form von Systemprogrammen wie z. B. Shell, Compiler und Editor.

Beispielsweise legt ein Anwendungsprogrammierer fest, welche Operationen aus der Programmierschnittstelle ein Anwendungsprogramm nutzen soll. Zum Editieren von Programmtexten kann ein Text-Editor verwendet werden, und dieser kann mit Maus und Tastatur als Benutzerschnittstelle gesteuert werden.

Die Hardware bietet dem Betriebssystem eine Menge von Maschinenbefehlen an, die *Instruction set architecture* (ISA), das Betriebssystem verwendet wiederum diese Instruktionen, um mit der Hardware zu kommunizieren.

3. Vernetzen

Ein dritter Grund für die rasante Verbreitung von Computern liegt darin, dass man sie zu *Netzwerken* zusammenschalten kann, die ganz neue Möglichkeiten eröffnen. So kann etwa ein Netz aus einfachen Personalcomputern den Mitarbeitern einer Abteilung den Zugriff auf gemeinsame Datenbestände erlauben und dabei viel preisgünstiger sein als ein Abteilungsrechner mit gleicher Gesamtleistung und einer entsprechenden Anzahl von Terminals. Ein weiteres Beispiel ist das *World Wide Web*, das den für den Einzelnen zugänglichen Informationsraum in einer Weise erweitert, die bis zum Ende der 1980er Jahre noch nicht vorstellbar war.

Kursinhalt

In diesem Kurs beschäftigen wir uns in den ersten beiden Kurseinheiten mit *Betriebssystemen*; die Kurseinheiten drei und vier befassen sich mit *Rechnernetzen*. Natürlich kann keines der beiden Themen in diesem Rahmen umfassend behandelt werden; unser Ziel ist es, Ihnen einige wesentliche Prinzipien und Grundtatsachen nahezubringen, die in der Praxis immer wieder auftreten und deswegen von allgemeinem Interesse sind. Wer diesen Stoff im späteren Studium vertiefen möchte, sei z. B. auf die Kurse *Betriebssysteme*, *Verteilte Systeme*, *Kommunikations- und Rechnernetze* und *Sicherheit im Internet* hingewiesen.

Vertiefung

vom Problem zur Computerlösung

An welchem Platz steht nun der Inhalt dieses Kurses im Gesamtkontext der Informatik? Stark vereinfacht kann man sagen, die Informatik beschäftigt sich damit, wie man von einem Problem der realen Welt zu einer Computerlösung kommt.

Problemspezifikation

Am Anfang wird im Dialog mit dem Anwender durch Abstraktion eine zunächst noch unscharfe Beschreibung des realen Problems gewonnen; sie muss dann weiter präzisiert werden, bis eine formale *Problemspezifikation* vorliegt, auf der dann das Pflichtenheft aufbauen kann. Dieser schwierige Prozess ist in der Informatik Gegenstand des *Software Engineering*.

[3] Wenn Benutzerschnittstellen auch aus graphischen Elementen bestehen, dann sprechen wir manchmal von *Benutzeroberflächen*.

Algorithmus

Anschließend wird ein *Algorithmus* zur Lösung des Problems entwickelt und auf Korrektheit und Effizienz geprüft. Ein wenig mathematisches Rüstzeug ist dabei unentbehrlich.

Es gibt wichtige Problembereiche, für die im Laufe der Zeit eigene algorithmische Techniken entstanden sind; Beispiele sind die Bereiche *Datenbanken und Informationssysteme*, *Künstliche Intelligenz*, *Computergraphik* und *kombinatorische Algorithmen*. Zu allen diesen Bereichen können Sie Wahlkurse der Informatik in Ihrem Studiengang belegen.

Programm
Programmierung

Aus dem Algorithmus entsteht dann ein *Programm*, zum Beispiel in Pascal, wie es im Kurs *Einführung in die imperative Programmierung* gelehrt wird, oder in Java, wie es der Kurs *Einführung in die Objektorientierte Programmierung* vermittelt.

Maschine

Wie aus einem für Menschen lesbaren Programmtext ein Programm in Maschinensprache (Instruktionen der ISA) wird, zeigt der Kurs *Übersetzerbau*. In den Kursen der *Technischen Informatik* wird erklärt, wie die Maschine, also die Hardware-Plattform, aufgebaut ist.

Programmausführung

Zu guter Letzt kommt dieser Kurs zum Tragen: Wir untersuchen hier, was geschieht, während Maschinenprogramme von Computern ausgeführt werden.

Zielgruppen

Wer hat mit Betriebssystemen und Rechnernetzen zu tun? Zwei Gruppen hatten wir bereits identifiziert:

Benutzer

- Wer als *reiner Benutzer* auf seinem Computer ausschließlich fertige Anwendungsprogramme laufen lässt, arbeitet gelegentlich mit der Benutzerschnittstelle des Betriebssystems, etwa um ein Programm zu starten, ein neues Verzeichnis anzulegen oder um nach einer Datei zu suchen.

Anwendungsprogrammierer

- Wer als *Anwendungsprogrammierer* selbst Software entwickelt, muss die Programmierschnittstelle des Betriebssystems kennen. Bei allen netzbasierten Anwendungen sind außerdem solide Kenntnisse über Rechnernetze vonnöten, zum Beispiel über Protokolle und den Client-Server-Betrieb.

Es lassen sich zumindest zwei weitere Berufsgruppen ausmachen, für die der Inhalt dieses Kurses ebenfalls interessant ist:

Systemadministrator

- Wer als *Systemadministrator* einzelne Computer einrichtet und an ein vorhandenes Netzwerk anschließt oder selbst ein kleines oder größeres Netzwerk einrichtet, betreibt und wartet, muss neben den Benutzer- und Programmierschnittstellen der Betriebssysteme der beteiligten Computer auch deren spezielle Kommandos für den Systemadministrator (Super-User) kennen, mit denen die Konfiguration des Rechners und des Netzwerks festgelegt wird. Benötigt werden auch Kenntnisse der internen Abläufe, um Performanzprobleme beheben zu können. Schließlich sind für die Wartung auch grundlegende Hardwarekenntnisse erforderlich.

- Wer schließlich selbst an der *Weiterentwicklung von Betriebssystemen* oder anderer Systemsoftware[4] mitarbeitet, muss detaillierte Kenntnisse über die interne Struktur des Betriebssystems und der Rechnerhardware besitzen.

Systementwickler

Natürlich lassen sich diese Rollen nicht scharf gegeneinander abgrenzen; wer etwa privat intensiv mit seinem PC arbeitet, ist oft Benutzer, Programmierer und Administrator in einer Person.

Bevor wir nun mit den Betriebssystemen beginnen, noch ein paar Vorbemerkungen. Ein Kurs über Betriebssysteme will in allgemeiner Form beschreiben, welche Aufgaben moderne Betriebssysteme erfüllen müssen, und Prinzipien zur Lösung dieser Aufgaben vorstellen. Er will *nicht* die Dokumentation eines konkreten Betriebssystems ersetzen[5]. Trotzdem wird in diesem Kurs gelegentlich ein reales Betriebssystem als Beispiel auftreten, und die Beispiele sollten für Sie zu Hause am Rechner nachvollziehbar sein, deshalb haben wir uns für Linux entschieden. Informationen zu Linux und Installationshinweise stehen zum Beispiel in [1, 2, 3, 4, 5].

Linux als Beispiel

Die Fachsprache der Informatik ist Englisch. Hilfetexte und Handbücher sind oft nur auf Englisch verfügbar. Wer Informatik studiert, sollte deshalb nach besten Kräften Englisch lernen.[6] Wir ergänzen ihn hier durch einschlägige Fachvokabeln aus den Bereichen Betriebssysteme und Rechnernetze, die meist in Klammern hinter den deutschen Begriffen aufgeführt werden.

In den nächsten Kapiteln finden Sie an verschiedenen Stellen Links und QR-Codes, die sie zu computerunterstützten *Übungsaufgaben* führen. Sie sind mit nebenstehendem Zeichen gekennzeichnet, das auch den Schwierigkeitsgrad angibt (1 = leicht, 5 = schwierig). Diese Übungsaufgaben geben Ihnen die Möglichkeit, den im Kurstext behandelten Stoff zu vertiefen und ihren Lernerfolg zu kontrollieren. Dazu geben Ihnen die computerunterstützten Übungsaufgaben Feedback zu Ihrer Lösung und Hinweise auf relevante Stellen im Kurstext. Wir empfehlen Ihnen nachdrücklich, die Übungsaufgaben aktiv zu bearbeiten. Sie sollten dazu auf der Grundlage des Kurstextes, ohne weitere Hilfsmittel benutzen zu müssen, in der Lage sein.

1.2 Die Hardwarekomponenten eines Computers

In der Einführung hatten wir festgestellt, dass das Betriebssystem eine Zwischenschicht ist, die die Benutzer und ihre Programme von der „nackten“

[4] Zur Systemsoftware zählen z. B. Debugger, Datenbankmanagementsysteme (DBMS), graphische Benutzeroberflächen, Compiler und Binder, Netzsoftware, aber auch das Betriebssystem selbst.

[5] Denn Dokumentationen veralten rasch, während allgemeine Prinzipien ihre Gültigkeit behalten.

[6] Solide Englischkenntnisse bilden eine Schlüsselqualifikation, die in zahlreichen Stellenausschreibungen verlangt wird. Sprachvermischungen, wie „Die CPU macht busy-waiting, bis ein interrupt gehandlet werden muss“ sind aber nicht so gern gesehen.

Hardware des Rechners trennt. Um die Funktionen des Betriebssystems richtig verstehen zu können, sollten wir uns deshalb eine ungefähre Vorstellung davon machen, wie die Hardware aufgebaut ist. Dazu dient dieser Abschnitt.

Wir beschränken uns dabei auf die klassische Architektur des von-Neumann-Rechners,[7] wie sie auch im Kurs *Einführung in die imperative Programmierung* beschrieben wird.

Programmieren durch physische Veränderung der Hardware

Um diese Architektur richtig würdigen zu können, muss man wissen, dass in der Anfangszeit die Rechner durch *physische Veränderung der Hardware* programmiert wurden, der legendäre ENIAC[8] etwa durch das Umstöpseln von Steckverbindungen.

von Neumanns Idee

Im Jahr 1945 hatte dann von Neumann eine bahnbrechende Idee: Das auszuführende Programm ist nicht mehr ein fester Bestandteil des Rechners; es wird vielmehr – genau wie die benötigten Daten – vor dem Programmlauf in den Speicher des Rechners geladen und hinterher wieder entfernt. Fest in den Rechner einbauen muss man also nur noch die Fähigkeit, ein beliebiges im Speicher befindliches Programm ausführen zu können – und hieran braucht man dann nie wieder etwas zu verändern!

universelle Maschine zur Ausführung von Programmen

Diese Vorstellung vom Computer als einer *universellen Maschine zur Ausführung von Programmen* wird Ihnen in der Theoretischen Informatik wiederbegegnen.

Bestandteile der Hardware

Die wesentlichen Bestandteile der Hardware eines modernen von-Neumann-Rechners sind daher

- der Prozessor (CPU = central processing unit),
- der Hauptspeicher (main memory) und
- die Ein-/Ausgabegeräte (I/O devices); dazu zählen typischerweise
  - Bildschirm (monitor), Maus (mouse) und Tastatur (keyboard), zusammen auch Terminal genannt,
  - Kommunikationsgeräte zum Anschluss an Rechnernetze, zum Beispiel Ethernet-Controller oder WLAN-Controller,
  - Festspeicher wie SSD (solid state drive) und Magnetplattenlaufwerk (HD, hard disk),
  - Wechselspeicher wie SD-Karte (secure digital memory card), USB-Stick, optische Medien wie CD und DVD oder früher Diskettenlaufwerk (floppy disk drive) und Bandlaufwerk (tape drive),
  - Drucker (printer),
  - Mikrofon, Kamera und diverse Spielecontroller (game controller).

In den nächsten beiden Abschnitten gehen wir auf einige dieser Hardwarebestandteile etwas näher ein.

[7] John von Neumann war einer der vielseitigsten Mathematiker des 20. Jahrhunderts. Er hat nicht nur das Bild des modernen Rechners geprägt, er war zum Beispiel auch ein Begründer der Spieltheorie.

[8] ENIAC, gebaut ab 1942, war ein Elektronenrechner, der mit 18.000 Röhren bestückt war, 30 Tonnen wog und einen derart hohen Stromverbrauch hatte, dass beim Einschalten das Licht in ganz Philadelphia dunkler geworden sein soll.

1.2.1 Prozessor und Hauptspeicher

Alle eigentlichen Berechnungen eines Computers finden in seinem *Prozessor* statt, genauer gesagt im Rechenwerk, das manchmal auch als ALU (arithmetic-logic unit) bezeichnet wird.

Aufbau des Hauptspeichers

Der Prozessor greift auf die Programme und die Daten zu, die sich zur Laufzeit im *Hauptspeicher* befinden. Dieser ist als eine lange Folge von gleich großen Speicherzellen organisiert, die einzeln adressiert werden können. *Zugreifen* heißt hier, dass das Datum aus einer Speicherzelle gelesen oder in sie geschrieben wird.

Wegen dieses *wahlfreien Zugriffs* hat sich für den Hauptspeicher auch die Bezeichnung RAM (random access memory) eingebürgert.[9]

In Pascal-Notation ist also der Hauptspeicher ein sehr langes

$$\text{array}\,[\,0\,..\,n-1\,]\ \text{of word},$$

Bytes und Bits

wobei ein Wort gerade der Inhalt einer Speicherzelle ist. Es besteht aus einem oder mehreren Bytes. Ein Byte enthält acht Bit, und ein Bit hat den Wert null oder eins. Die Speicherkapazität des Hauptspeichers beträgt daher n mal die Wortlänge. Sie wird in Kilobyte (KByte), Megabyte (MByte) oder Gigabyte (GByte) angegeben, wobei mit Zweierpotenzen gerechnet wird:[10]

$$1\ \text{GB} = 2^{10}\ \text{MB} = 2^{20}\ \text{KB} = 2^{30}\ \text{Byte}.$$

Zahlen und Befehle

Bei der von-Neumann-Architektur enthält der Hauptspeicher Daten und Programme. Entsprechend kann ein Wort die Binärdarstellung einer Zahl sein, der Code für einen Prozessorbefehl oder die Adresse einer anderen Speicherzelle. Einem Wort selbst kann man nicht ansehen, was es darstellt. Trotzdem sind Verwechselungen ausgeschlossen, denn der Prozessor „weiß“ bei jedem Zugriff auf eine Speicherzelle, ob darin eine Zahl oder ein Befehl steht.

Um ein Wort aus einer Speicherzelle zu holen, hat die CPU einige wichtige *Register*, die als Zwischenspeicher verwendet werden:

- Der *Befehlszähler* (*Program Counter, PC*) enthält die Adresse derjenigen Speicherzelle, in der der als nächstes auszuführende Befehl steht. — Befehlszähler
- Das *Befehlsregister* (*Instruction Register IR*) speichert den aktuellen Befehl, der gerade verarbeitet bzw. ausgeführt wird. — Befehlsregister
- Das *Speicheradressregister* (*Memory Address Register MAR*) enthält die Adresse derjenigen Speicherzelle, die als nächstes gelesen oder beschrieben werden soll. — Speicheradressregister

[9] Das Wort *random* bedeutet *zufällig*, aber dem Zufall bleibt beim Speicherzugriff nichts überlassen; „random access“ meint, dass die Zugriffsreihenfolge nicht im voraus bekannt ist und jede Speicherzelle mit ihrer Adresse direkt angesprochen werden kann. Daraus folgt, dass die Zeit für den Zugriff auf eine Speicherzelle eine Konstante ist, die nicht von der Anzahl der gespeicherten Elemente, der Speichergröße oder der Adresse abhängt.

[10] Wir verwenden im Kurs noch nicht die Schreibweise der *IEC-Präfix-Notation*: 1 MiByte = 1 Mebibyte = 2^{10} KiByte = 2^{10} Kibibyte = 2^{20} Byte, was eigentlich richtig wäre, denn unter Umständen wird 1 Kilobyte als Dezimalnotation (1000 B) verstanden. Wenn z. B. von Festspeicherherstellern Speicherkapazitätsangaben gemacht werden, dann sind sie meist dezimal gemeint, mit dem vielleicht erwünschten Effekt, dass dabei etwas größere Zahlen herauskommen.

Programmstatuswortregister

- Das *Programmstatuswortregister* enthält verschiedene Bits, die zeigen, ob das gerade ausgeführte Programm privilegierte Befehle benutzen darf oder ob die CPU eine Unterbrechung bearbeiten will.

Akkumulator

- Der *Akkumulator* ist eines von vielen *Datenregistern* und speichert ein Zwischenergebnis einer Berechnung.

Instruktionszyklus

Die CPU arbeitet Befehle in Zyklen ab, dem sogenannten *Instruktionszyklus*. Eine Befehlsausführung kann in zwei Phasen aufgeteilt werden:

Holphase

1. In der *Holphase* (*fetch stage*) wird das Speicheradressregister mit dem Wert des Befehlszählers belegt und die Adresse an den Adressbus weitergegeben. Der Inhalt der Speicherzelle mit dieser Adresse wird in das Befehlsregister geladen. Zum Abschluss wird in dieser Phase der Befehlszähler um Eins erhöht.

Ausführungsphase

2. In der *Ausführungsphase* (*execution stage*) wird der in das Befehlsregister geladene Befehl ausgeführt. In dieser Phase können auch weitere Daten oder Adressen von Speicherzellen geholt werden.

Übungsaufgabe 1.1 Identifizieren Sie den korrekten Ablauf des Instruktionszyklus...

`https://e.feu.de/1801-instruktionszyklus`

Schauen wir dem Prozessor einmal bei der Arbeit zu! Zur Illustration dient Abbildung 1.2. Sie zeigt einen Hauptspeicher der Kapazität 64 KByte, der aus 2^{16} Speicherzellen der Wortlänge 1 Byte besteht, und einige typische Bestandteile einer CPU, wie sie zum Beispiel beim Intel 8080 anzutreffen waren.

Das *Befehlszählregister* in der CPU enthält die Adresse – also die Binärdarstellung[11] des Index – von derjenigen Speicherzelle, in der der Code des nächsten auszuführenden Befehls steht. In Abbildung 1.2 ist das die Zelle mit dem Index 4; die Adresse als 16-Bit-Zahl lautet also

0000000000000100.

Adressbus

Datenbus

Um auf diese Speicherzelle zugreifen zu können, wird die Adresse in das Speicheradressregister übertragen und gelangt auf den *Adressbus*. Hierdurch wird die Zelle des Hauptspeichers mit Index 4 angesprochen, und ihr Inhalt wird über den *Datenbus* in das Speicherinhaltsregister des Prozessors übertragen. Weil durch die vorderen Bits des Wortes 01100111 der Prozessor erkennt, dass das Wort einen Befehl darstellt, wird es in das Befehlsregister kopiert; dort wird der Befehl decodiert und interpretiert. Die Holphase ist fertig.

Angenommen, der auszuführende Befehl lautet

$$A := A + (HL).$$

[11] Die Binärdarstellung einer Zahl ist Gegenstand von Übungsaufgabe 1.2.

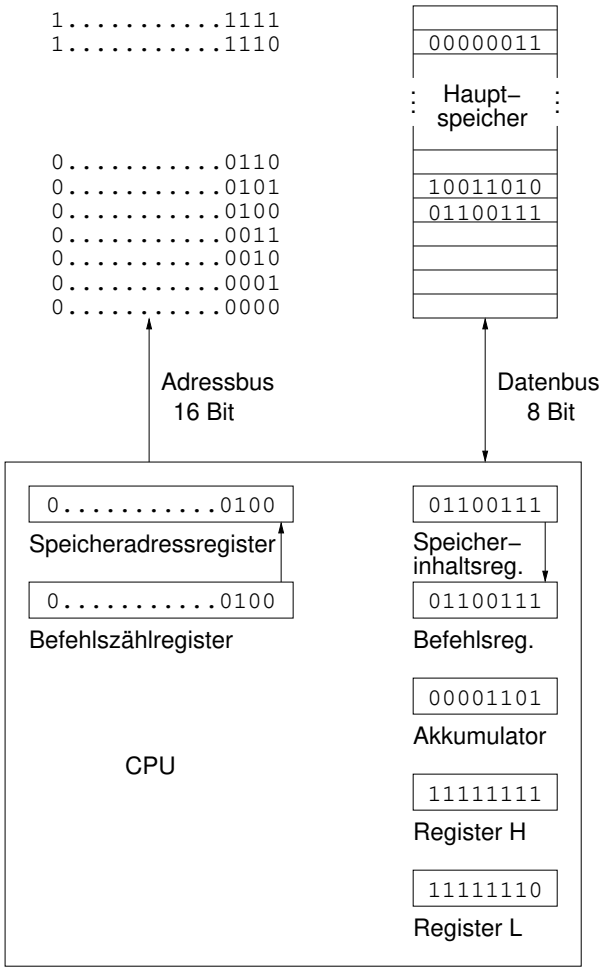

Abbildung 1.2: Der Prozessor greift über Daten- und Adressbus auf den Hauptspeicher zu.

Das bedeutet: Der Inhalt des Akkumulators, eines speziellen 8-Bit-Registers der CPU für die Speicherung eines Zwischenergebnisses, soll um die Zahl erhöht werden, die in der Speicherzelle steht, deren Adresse sich ergibt, wenn man die Worte in den Registern H (high) und L (low) hintereinanderschreibt; man spricht hierbei von *indirekter Adressierung*. Im Beispiel ergibt sich hierdurch die Adresse

Indirekte Adressierung

$$1111111111111110;$$

sie wird in das Speicheradressregister übertragen, und ein weiterer Speicherzugriff wird ausgelöst, der das Wort 00000011 in das Speicherinhaltsregister bringt. Aus dem Zusammenhang ist klar, dass es sich hierbei um die Binärdarstellung einer Zahl handeln muss, denn das Wort tritt ja als Operand einer Addition auf. Diese Addition kann nun ausgeführt werden; im Akkumulator steht danach die Binärdarstellung der Zahl $13 + 3 = 16$. Nun könnte zum Beispiel anschließend das Ergebnis der Addition in eine Hauptspeicherzelle zurückgeschrieben werden. Die Ausführungsphase ist zu Ende und ein Befehlsausführungszyklus ist fertig.

Programmzähler
Sprungbefehl

Der Inhalt des Befehlszählers wird am Ende der Holphase immer um Eins erhöht. Durch dieses Hochzählen um Eins erklärt sich auch der Name; man findet auch die Bezeichnung *Programmzähler* (program counter). Ein *Sprungbefehl* hingegen setzt den Befehlszähler auf einen bestimmten Wert, dies entspricht der Aktion „verzweige im Programmablauf zu einer bestimmten Adresse“.

Übungsaufgabe 1.2 Die Binärdarstellung einer natürlichen Zahl $n \geq 1$ ist die Folge $a_k a_{k-1} \ldots a_1 a_0$ von Nullen und Einsen a_i, für welche

$$n = a_k \cdot 2^k + a_{k-1} \cdot 2^{k-1} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

gilt und $a_k = 1$; führende Nullen lässt man also fort. Welche Zahl n wird durch 10011101 dargestellt? Und bestimmen Sie die Binärdarstellung von 160.

`https://e.feu.de/1801-binaerdarstellung`

Übungsaufgabe 1.3 Beim Prozessor 68000 von Motorola beträgt die Wortlänge 2 Byte. Wie breit muss der Adressbus sein – das heißt: aus wievielen Bits bestehen die Adressen – bei einer Hauptspeicherkapazität von 16 MByte?

`https://e.feu.de/1801-addressbusbreite`

Takt
Gigahertz

Die CPU arbeitet *getaktet*. Ein Maschinenbefehl erfordert bei der Ausführung ein oder mehrere Takte. Einfache Maschinenbefehle, wie etwa das Kopieren einer Adresse vom Befehlszählerregister ins Speicheradressregister können in einem Takt ausgeführt werden. Als *Taktfrequenz* bezeichnet man die Anzahl der Takte pro Sekunde; eine Frequenz von z. B. 1 GHz (Gigahertz) bedeutet einen Takt von einer Nanosekunde bzw. eine Milliarde Takte pro Sekunde.

Zugriffszeit

Auch das Heraufzählen des Befehlszählregisters lässt sich in einem einzigen Arbeitstakt erledigen. Dagegen nimmt ein Hauptspeicherzugriff viele Arbeitstakte in Anspruch.[12] Denn einerseits sind mehrere Maschinenbefehle auszuführen, wie wir am Beispiel des Befehls A := A + (HL) oben gesehen haben. Andererseits vergeht eine gewisse Zeit zwischen dem Aufbringen einer Adresse auf den Adressbus und der Ankunft des zugehörigen Speicherzelleninhalts über den Datenbus. Diese Zeitspanne hängt von der physikalischen Beschaffenheit des Hauptspeichers und der Geschwindigkeit des Datenbusses ab; sie wird als *Zugriffszeit* auf den Hauptspeicher bezeichnet.

[12] Es gibt auch andere Prozessorbefehle, die mehr als einen Arbeitstakt erfordern. Manche Hersteller geben deshalb neben der Taktfrequenz auch die mittlere Anzahl von Befehlen an, die pro Sekunde abgearbeitet werden kann. Gemessen wird diese Zahl in MIPS (million instructions per second).

1.2.2 Cache

Schnelle Hauptspeicherchips kosten pro MByte Speicherkapazität viel mehr als langsame Chips. Aus Kostengründen wird deshalb oft neben dem normalen Hauptspeicher ein kleinerer, aber schnellerer Zwischenspeicher verwendet, der meist als *Cache* bezeichnet wird.[13] Häufig benutzte Daten werden vorübergehend vom Hauptspeicher in den Cache kopiert. Wenn Daten benötigt werden, schaut man zunächst im Cache nach. Findet man dort eine Kopie der gesuchten Daten, so verwendet man sie und braucht keinen Hauptspeicherzugriff auszuführen. Wird man aber im Cache nicht fündig, ist ein Zugriff auf den Hauptspeicher unvermeidlich; in diesem Fall legt man eine Kopie der Daten im Cache ab, weil man davon ausgeht, dass diese Daten bald noch einmal benötigt werden. Dieser einfache *Cache-Algorithmus* spielt eine wichtige Rolle; er wird uns in Abschnitt 1.2.3 wiederbegegnen.

Cache

Cache-Algorithmus

Für das Betriebssystem stellen sich hier zwei wichtige Aufgaben:

1. **Cache-Management**
   Wenn Daten im Cache abgelegt werden sollen und dort kein freier Platz mehr vorhanden ist, müssen alte Daten überschrieben werden. Welche Daten soll man dafür opfern?[14]

   Cache-Management

2. **Cache-Konsistenz**
   Angenommen, der Wert einer Variablen soll verändert werden. Wenn diese Änderung nur an der Kopie im Cache vollzogen wird, so besteht anschließend ein Unterschied zwischen dem Original im Hauptspeicher und der Kopie im Cache. Das Betriebssystem muss dafür sorgen, dass sich hieraus keine Fehler ergeben. Dieses Problem ist besonders gravierend bei *Multiprozessorsystemen*, bei denen jeder Prozessor über einen eigenen Cache verfügt.

   Cache-Konsistenz

Welchen Effizienzgewinn die Verwendung eines schnellen Zwischenspeichers bringt, hängt zum einen von der Schnelligkeit der Hardware und vom Cache-Management ab, im Einzellfall aber auch davon, wie ein konkretes Programm auf seine Daten zugreift.

Übungsaufgabe 1.4 Angenommen, ein Schreib-/Lesezugriff auf den Cache ist neunmal so schnell wie ein Zugriff auf den Hauptspeicher. Welchen Zeitvorteil ergibt die Verwendung eines Cache nach dem oben beschriebenen Algorithmus, wenn bei 80 % aller Zugriffe die gesuchten Daten im Cache gefunden werden?

`https://e.feu.de/1801-caching`

[13] Das Wort Cache klingt im Englischen ähnlich wie *cash* = Bargeld. Auch wenn ein schneller Cache viel Geld kostet, sollte man die beiden Wörter nicht verwechseln.

[14] Erst durch diese Festlegung wird ein Cache-Algorithmus vollständig bestimmt. Man kann zum Beispiel die Strategie FIFO (first-in, first-out) anwenden und stets diejenigen Daten überschreiben, die schon am längsten im Cache stehen.

Hauptspeicher: knapp und flüchtig

Mit Hauptspeicher (und Cache) allein kann ein Rechner nicht auskommen, denn

1. selbst wenn der Hauptspeicher pro MByte preiswerter als der Cache ist, kann man seine Kapazität aus Kostengründen nicht so groß auslegen, dass alle benötigten Programme und Daten darin Platz finden;
2. beim Abschalten des Rechners oder bei einem Stromausfall geht der Inhalt des Hauptspeichers verloren.

Außerdem braucht man Geräte, mit deren Hilfe der Rechner Programme und größere Datenmengen mit seiner Umwelt austauschen kann. Aus diesem Grund besitzen die meisten Computer außer dem Hauptspeicher, der auch als *Primärspeicher* bezeichnet wird, noch *Sekundärspeicher* und *Tertiärspeicher*. Mehr hierzu finden Sie in den nächsten Abschnitten.

1.2.3 Sekundärspeicher

Der Hauptspeicher wird auch als *flüchtiger Speicher* bezeichnet, d. h. die Daten im Speicher gehen beim Ausschalten oder bei einem Neustart verloren.

Der Rechner kann aber erst anfangen, Programme auszuführen, wenn sie auch im Hauptspeicher vorliegen. Dafür benötigt man einen Speicher (*Festspeicher*), der die Programme und Daten dauerhaft speichern kann, die dann in den Hauptspeicher geladen werden. Die Aufgabe für die dauerhafte Speicherung übernimmt ein *Sekundärspeicher* (secondary memory). Die gebräuchlichsten Typen von Sekundärspeicher sind zur Zeit die *Magnetplatte* und der *Flashspeicher*.

Festspeicher

Sekundärspeicher

Magnetplatte

Flashspeicher

1.2.3.1 Festplatte

Eine Magnetplatte ist auf beiden Seiten mit einer magnetisierbaren Oberfläche beschichtet und dreht sich mit rund einhundert Umdrehungen pro Sekunde. Früher gab es Wechselplatten, heute sind die Magnetplatten meist fest in ihr Laufwerk eingebaut. Bei manchen Plattenlaufwerken sind mehrere Platten übereinander angebracht, die von einer gemeinsamen Spindel gedreht werden; siehe Abbildung 1.3. Auf jeder Seite einer Platte gleitet auf einem dünnen Luftkissen ein Schreib-/Lesekopf. Er wird von einem Arm geführt, der an einer Welle befestigt ist. Durch Verdrehen der Welle werden alle Köpfe simultan positioniert.

Festplatte

Diese Anordnung mag an altmodische Plattenspieler erinnern, es bestehen aber zwei wesentliche Unterschiede zur Vinyl-Schallplatte: Bei der Festplatte wird die Information magnetisch gespeichert, und die Daten sind nicht spiralförmig auf der Festplatte angeordnet; vielmehr ist jede Oberfläche einer Platte in einige tausend kreisförmige *Spuren* gleicher Breite unterteilt, die wiederum aus einigen hundert *Sektoren* bestehen. Die jeweils übereinanderliegenden Spuren bilden einen *Zylinder*. Die weiter außen liegenden Spuren können mehr Sektoren enthalten als die inneren.

Spuren, Sektoren, Zylinder

Bei starken Erschütterungen kann es vorkommen, dass ein Schreib-/Lesekopf die Plattenoberfläche im Betrieb berührt und sie beschädigt. Bei

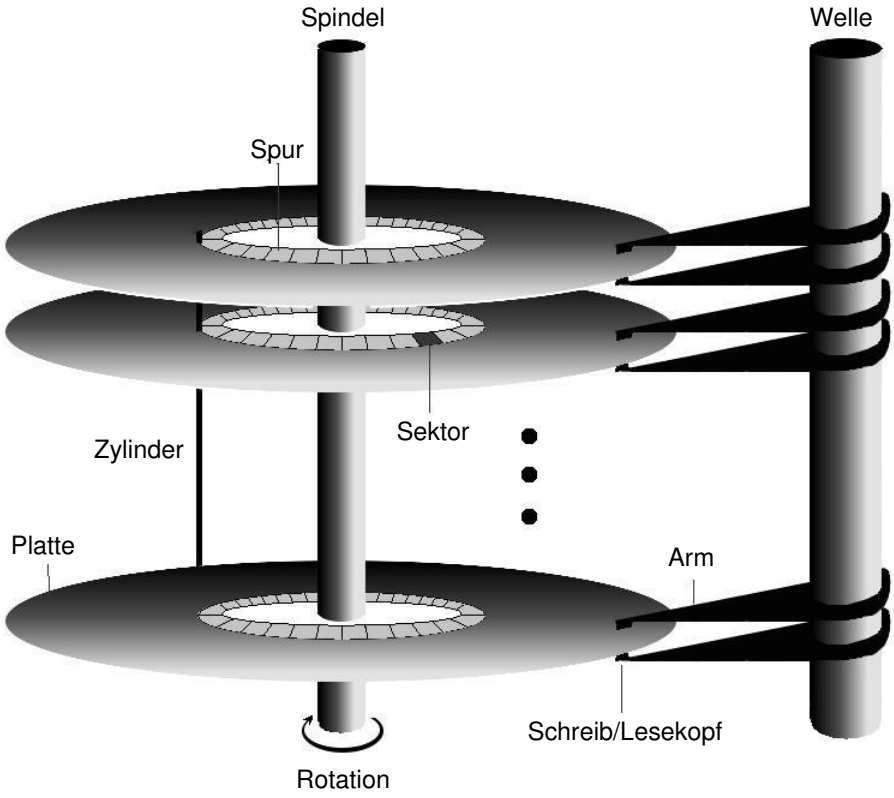

Abbildung 1.3: Ein Plattenstapel mit Schreib-/Leseköpfen.

einem solchen *head crash* können Daten verlorengehen. Um dieses Risiko zu verringern, sind Magnetplatten meist in staubdicht verschlossenen Gehäusen untergebracht, und der Arm wird in einer sicheren Ruhestellung geparkt, wenn keine Aufträge vorliegen oder wenn eine zu starke Bewegung (z. B. fallend) registriert wird.

wahlfreier Zugriff

Die Magnetplatte ermöglicht *wahlfreien Zugriff* auf einzelne Sektoren, d. h. jeder Sektor wird mit einer Adresse angesprochen. Um den Inhalt eines bestimmten Sektors zu lesen, muss zunächst der Kopf durch Verdrehen der Welle auf den richtigen Zylinder gebracht werden. Die Zeit für die Positionierung des Kopfes nennt man *Positionierungszeit.* Dann wird abgewartet, bis der gesuchte Sektor am Kopf vorbeiläuft (*Latenzzeit*). Schließlich kann der Sektorinhalt gelesen werden. Die gelesenen Daten kommen zunächst in einen Pufferspeicher (buffer). Die Zeit, die hierfür benötigt wird, nennt man *Übertragungszeit.* Schreibzugriffe funktionieren entsprechend. Die *Zugriffszeit* auf einen Sektor ist definiert als die Summe von Positionierungs-, Latenz- und Übertragungszeit.

Positionierungszeit

Latenzzeit

Puffer

Übertragungszeit

Am längsten dauert gegenwärtig noch die Positionierung des Schreib-/Lesekopfes auf den richtigen Zylinder. Die Summe aus Positionierungs- und Latenzzeit nennt man *Suchzeit* (seek time). Die Zugriffszeit auf eine Magnetplatte ist deshalb sehr viel länger – etwa tausend mal – als die Zeit, die beim Zugriff auf eine Zelle im Hauptspeicher vergeht. Im Unterschied zum Zugriff auf eine Zelle im Hauptspeicher ist die Zugriffszeit auf einen Sektor keine Konstante, die nur von der physikalischen Beschaffenheit der Magnetplatte abhängt, da die Suchzeit von der aktuellen Position des Lesekopfes abhängt.

Suchzeit

Zugriffszeit

Suchzeit verringern

Die Suchzeit für die Kopfpositionierung entspricht in etwa der Distanz, die der Kopf zurücklegt. Diese Distanz wird durch die Anzahl der Spuren gemessen, über die der Kopf bewegt werden muss. Folgende Maßnahmen helfen, die Suchzeit zu verringern:

- zusammengehörende Information sollte möglichst in benachbarten Sektoren und Zylindern gespeichert werden;
- wenn bei einem Plattenlaufwerk mehrere Schreib-/Leseaufträge gleichzeitig vorliegen, sollte eine günstige Bearbeitungsreihenfolge gewählt werden (disk scheduling).

Auch diese Aufgaben werden vom Betriebssystem erledigt; näheres hierzu folgt in Abschnitt 1.3.

Übungsaufgabe 1.5 Wir betrachten ein Plattenlaufwerk mit nur einer einzelnen Platte; die Spuren sind mit 0 beginnend von außen nach innen durchnummeriert. Wenn dafür nun mehrere Aufträge vorliegen, kann man sie nach folgenden Strategien abarbeiten:

- FCFS (first-come, first-served) bearbeitet die Aufträge in der Reihenfolge ihres Eingangs.
- SSTF (shortest-seek-time-first) bearbeitet jeweils denjenigen Auftrag als nächsten, dessen Spur der momentanen Position des Schreib-/Lesekopfs am nächsten liegt.
- SCAN bewegt den Kopf abwechselnd von außen nach innen und von innen nach außen über die gesamte Platte und führt dabei die Aufträge aus, deren Spuren gerade überquert werden.

Angenommen, es sind Zugriffsaufträge für Sektoren in den Spuren

32, 185, 80, 126, 19, 107

in dieser Reihenfolge eingegangen. Der Kopf steht anfangs auf Spur 98 und bewegt sich im Falle der SCAN-Strategie gerade nach außen. Welche Distanz muss der Kopf bei Anwendung der drei Strategien jeweils insgesamt zurücklegen, bis alle Aufträge erledigt sind?

`https://e.feu.de/1801-plattenzugriff`

parity bit

Zwar dauert ein Zugriff auf die Magnetplatte sehr viel länger als ein Hauptspeicherzugriff, aber dafür liefert er ein sehr viel größeres Datenvolumen: Ein Sektor enthält zwischen 0,5 und 4 KByte Nutzinformation, zusätzlich sind Sektor- und Zylindernummer und Hilfsdaten für die Fehlerkorrektur gespeichert. Zu diesem Zweck kann man sogenannte *parity bits* einführen, deren

Wert die Summe[15] der Werte bestimmter Datenbits sind. Beim Schreiben bzw. Lesen der Nutzinformation eines Sektors werden diese Summen berechnet und in die Prüfbits eingetragen bzw. mit den dort gespeicherten Werten verglichen. Ergeben sich Abweichungen, so ist ein Fehler aufgetreten. Bei Verwendung geeigneter Codes kann man feststellen, welches Bit einen falschen Wert hat und den Fehler automatisch korrigieren. Die Nutzinformation eines Sektors wird als ein *Block* bezeichnet.

Block

Um zeitaufwendige Plattenzugriffe nach Möglichkeit einzusparen, kann man einen *Cache* im Hauptspeicher einrichten, in dem einzelne Blöcke oder die Inhalte ganzer Spuren gespeichert werden, in der Annahme, dass man sie in Kürze noch einmal benötigt. Das Prinzip ist dasselbe wie in Abschnitt 1.2.2.

Cache für Platten

Manche Computer verfügen über eine sogenannte *RAM-disk*. Hierunter versteht man einen fest reservierten Teil des Hauptspeichers, der von Anwenderprogrammen und Anwendern wie eine Magnetplatte benutzt werden kann. Zum Beispiel werden Befehle zur Dateiverarbeitung, wie wir sie in Abschnitt 1.6 kennenlernen werden, in entsprechende Speicherzugriffe übersetzt. Das geht beinahe so schnell wie bei „reinen“ Hauptspeicherzugriffen. Ein Nachteil: Auch bei der RAM-disk sind die Daten nach Abschalten des Stroms verschwunden.

RAM-disk

1.2.3.2 Flashspeicher

Der *Flashspeicher* ist ein rein elektronischer Speicher der Art EEPROM (electrically erasable programmable read only memory). Ein EEPROM erlaubt es, den Inhalt von Speicherzellen elektrisch zu löschen und wieder zu beschreiben. Flashspeicher besitzt keine beweglichen Teile wie Magnetplatten und ist widerstandsfähig gegen Erschütterung. Daher wurden Flashspeicher zuerst für mobile Geräte und in anspruchsvollen Umgebungen eingesetzt, inzwischen wegen ihrer Geschwindigkeitsvorteile aber auch in jedem beliebigen Computer. In Allgemeinen werden sie als *solid state disk* (SSD) bezeichnet. Es gibt zwei Varianten von Flashspeichern, nähmlich NOR und NAND. Beide Varianten wurden von dem japanischen Elektroingenieur Fujio Masuoka in den 1980er Jahren erfunden.

solid state disk

Ein Flash-Chip besteht aus einer Anzahl von sogenannten *Erase Blocks*. Jeder Block hat die Größe 128 KByte oder 256 KByte. Bei NOR besteht ein Block aus Daten-Bits, die einzeln gelesen und geschrieben werden können, d. h. jedes Daten-Bit kann einzeln von 1 auf 0 gesetzt werden. Bei NAND besteht ein Block aus einer Anzahl von sogenannten *Seiten*. Eine Seite enthält nicht nur Daten-Bits, sondern auch einen Out-Of-Band-Bereich(OOB) für die Fehlerbehandlung und die Kennzeichnung einer Beschädigung der Seite. Die Daten in einem Block können hier nur *seitenweise* gelesen und geschrieben werden. Die *Löschen-Operation* muss bei beiden Flash-Arten auf einem *kompletten Block* ausgeführt werden.

Erase Blocks

Seiten

Der Nachteil bei EEPROMs ist, dass eine Seite erst wieder geschrieben werden kann, nachdem sie gelöscht wurde. Das Löschen einer Seite bedeutet

[15] Beim Addieren von Bits wird modulo 2 gerechnet. Anders ausgedrückt: Eine Summe von Nullen und Einsen hat den Wert eins, wenn die Anzahl der Einsen ungerade ist, und sonst den Wert Null.

das Löschen des kompletten Blocks, in dem sich die Seite befindet. Eine Löschoperation setzt alle Bits des Blocks auf den Wert 1. Also kann eine Schreib-Operation den Zustand eines Bits nur von 1 auf 0 setzen. Bevor ein Block gelöscht wird, müssen zuerst die noch gültigen Seiten des Blocks auf andere, freie Seiten kopiert werden.

wear out

Ein Block kann nur eine bestimmte Anzahl von Löschoperationen vertragen, typischerweise zwischen 100 000 und einer Million, danach ist der Block abgenutzt (*wear out*) und kann nicht mehr für die Speicherung von Daten weiter verwendet werden. Deshalb benötigt ein Flashspeicher eine Organisationsstruktur, um zu wissen, welche Seiten und wie viele Seiten in einem Block gültig sind und wie oft die Löschoperation auf jedem Block stattgefunden hat. Diese Information ist wichtig für die Entscheidung, welcher Block als nächster gelöscht werden soll.

Auf der Magnetplatte und dem Flashspeicher gespeicherte Programme und Daten sind zwar gegen Stromausfall geschützt; trotzdem sind Platten und Flash-Speicher für die *Langzeitarchivierung* aus folgenden Gründen nicht gut geeignet:

- Magnetplatten sind für wahlfreien Zugriff konzipiert und deshalb pro MByte Kapazität zu teuer, um darauf Informationen abzulegen, auf die nur selten zugegriffen wird.
- Sie sind oft im selben Gehäuse untergebracht wie Prozessor und Hauptspeicher und erlauben es deshalb nicht, Informationen an getrenntem Ort zu verwahren oder auf andere Rechner zu transferieren, wie man es für eine Sicherungskopie benötigt.
- Die Blöcke eines Flashspeichers können nur begrenzt oft gelöscht werden. Regelmäßiges Sichern erfordert daher regelmäßigen Austausch des Speichermediums.

Aus diesen Gründen gibt es neben dem Sekundärspeicher auch *Tertiärspeicher*; damit beschäftigt sich der folgende Abschnitt.

1.2.4 Tertiärspeicher

Das wesentliche Merkmal von Tertiärspeichern ist, dass sich der Datenträger preiswert herstellen und leicht vom Rechner entfernen lässt. Sie eignen sich gut für den Transfer von Daten zwischen nicht vernetzten Rechnern.

Diskette

Bis ca. 2000 waren *Disketten* weit verbreitet, besonders bei PCs. Danach wurden Diskettenlaufwerke aber immer seltener in neue Computer eingebaut, heute sind sie praktisch verschwunden. Ihre Rolle als bequem transportierbares Speichermedium nahmen zunächst die CDs und DVDs ein, die aber wegen der eingeschränkten Wiederbeschreibbarkeit nicht wirklich funktionsgleich sind, und dann die USB-Sticks; siehe dazu weiter unten in diesem Abschnitt.

Diskettenlaufwerke sind im Prinzip ähnlich den Festplattenlaufwerken, sie sind aber einfacher aufgebaut. Auch die Diskette bietet wahlfreien Zugriff;

dem im Verhältnis zur Magnetplatte deutlich geringeren Preis stehen aber eine erheblich längere Zugriffszeit und ein viel kleineres Datenvolumen (nur ca. 1 MByte) gegenüber.

Magnetband

Für die *Langzeitarchivierung* von Daten und Programmen werden gern *Magnetbänder* verwendet. Ihre Kapazität ist sehr viel größer als die einer Magnetplatte, da auch die magnetisierbare Oberfläche größer ist. Wie im Audiobereich gibt es Bandspulen und Kassetten, die leichter zu wechseln sind. Auch Videokassetten finden Verwendung. Die Daten werden blockweise aufgezeichnet. Im Gegensatz zu den bis jetzt besprochenen Speicherarten bieten Magnetbänder aber nur *sequentiellen Zugriff*: Man kann nur auf die beiden Blöcke direkt zugreifen, die sich gerade vor oder hinter dem Schreib-/Lesekopf befinden. Will man andere Blöcke lesen, muss das Band erst vor- oder zurückgespult werden, was zu extrem langen Zugriffszeiten führt. Im Unterschied zur Magnetplatte brauchen beim Magnetband die Blöcke nicht gleich lang zu sein. Weil auf Bändern oft keine Dateisysteme angelegt werden, kann es Mühe bereiten, ein fremdes Band richtig zu lesen.

sequentieller Zugriff

In den Rechenzentren früherer Jahre war die Bedienmannschaft (Operator) dafür verantwortlich, bei Bedarf die benötigten Bänder oder Wechselplatten in die Laufwerke einzulegen. Diese Aufgabe kann heute von Robotern übernommen werden. Oft fasst man einige hundert Kassetten zu einer sogenannten *juke box*[16] zusammen, die mit einer Wechselmechanik versehen ist.

juke box

CD-ROM
DVD

Als Ersatz für die Disketten kamen zunächst die *CD-ROMs* (compact disks, kurz CDs) und dann die *DVDs* (digital versatile disks) zum Einsatz. Es handelt sich um 1,2 mm dicke Scheiben aus transparentem Polycarbonat mit 12 cm Durchmesser, die Speicherkapazität beträgt bei CDs mindestens 650 MByte und bei DVDs mindestens 4,38 und maximal 8,5 GByte. Entsprechende Laufwerke wurden seit ca. 1995 in praktisch alle Computer eingebaut, kommerzielle Software wurde oft auf CDs oder DVDs ausgeliefert, die bei hoher Kapazität extrem preisgünstig herzustellen und leicht zu transportieren sind. Die Daten werden optisch gespeichert und von einem Laser abgetastet.

Zu den nur lesbaren CD-ROMs und DVD-ROMs kamen dann auch schnell die einmal beschreibbaren CD-Rs und DVD-Rs sowie die löschbaren und deshalb mehrfach wiederbeschreibbaren Varianten CD-RW (compact disk – read/write) und DVD-RW, durch die die Anwendungsmöglichkeiten dieser Speichertechnik stark vergrößert wurden. Trotzdem sind diese Medien kein vollwertiger Ersatz für die Disketten, weil nicht beliebige Sektoren schnell gelöscht und überschrieben werden können. Die Zugriffszeiten für CDs und DVDs variieren je nach Umdrehungsgeschwindigkeit und Zugriffsart, sind aber jedenfalls wesentlich länger als die von Magnetplatten und kürzer als die von Disketten.

Blu-ray Disk
HD DVD

Die optischen Speichermedien *Blu-ray Disk* (kurz BD) und *HD DVD* wurden als Nachfolger der DVD entwickelt. Die beiden Formate sind technisch sehr ähnlich, den *Formatkrieg* hat schließlich die BD gewonnen. Die Speicher-

[16] Der Name *juke box* erinnert an die Musik-Boxen, die Vinyl-Schallplatten wechseln können.

kapazität der Blu-ray Disk beträgt mindestens 25 GByte, die Datenübertragungsrate ist viermal so hoch wie die einer DVD bei gleicher Umdrehungszahl, beträgt also mindestens 4,5 MByte pro Sekunde bei sogenannter „einfacher“ und 36 MByte pro Sekunde bei „achtfacher“ Geschwindigkeit.

Externe Festplatten

Externe Festplatten sind handelsübliche Festplatten, die statt direkt in den Computer in ein besonderes, meist recht kleines Gehäuse eingebaut und über eine Standardschnittstelle wie z. B. USB, Firewire oder eSATA angeschlossen werden. Oft erfolgt auch die Stromversorgung über diese Schnittstelle, so dass die Festplatte sehr leicht angeschlossen, beschrieben, abgezogen, transportiert und woanders wieder angeschlossen werden kann. Die Speicherkapazität ist im Vergleich zu CDs und DVDs sehr groß, und die Zugriffszeiten sind je nach Schnittstelle fast so kurz wie die von internen Festplatten.

Flash

Noch preisgünstiger als externe Festplatten sind *Flash*-basierte Speicher. Hier handelt es sich um Speicherchips, die Daten permanent, also auch ohne Stromversorgung, speichern, und die auch schnell überschrieben werden können. Solche Speicher existieren einerseits in der Form von sogenannten *USB-Sticks*, d. h. direkt an einen USB-Stecker angeschlossene Speicherchips in einem sehr kleinen Gehäuse, und andererseits als noch kleinere *Speicherkarten*, die in Digitalkameras und Mobiltelefonen eingesetzt werden. Kartenschächte zur Aufnahme dieser Speicherkarten werden in viele Rechner eingebaut.

USB-Sticks

Speicherkarten

Daten sichern!

Zum Abschluss unserer Besprechung der verschiedenen Speicherarten möchte ich einen dringenden Appell an Sie richten: Sichern Sie Ihre Programme und Daten regelmäßig, und nutzen Sie dabei die Möglichkeiten, die Ihnen die verschiedenen Speichertypen bieten!

Konkret bedeutet das: Wer interaktiv an einem längeren Text oder an einem Progammierprojekt arbeitet, sollte regelmäßig den aktuellen Stand auf der Festplatte sichern.[17] Ansonsten kann es passieren, dass bei einem Stromausfall oder bei einem Rechnerabsturz die Arbeit einiger Stunden verloren geht – vielleicht keine Katastrophe, aber doch höchst ärgerlich.

Speicherung auf einer Festplatte allein genügt aber nicht, denn bei einem Laufwerksdefekt kann der Platteninhalt verloren sein. Verschwindet dabei zum Beispiel kurz vor dem Abgabetermin Ihre Abschlussarbeit, so kann das durchaus katastrophale Folgen haben. Deshalb muss man regelmäßig von wichtiger Information Sicherheitskopien (backups) anlegen, sei es nun auf Diskette, Magnetband, CD, externer Festplatte, Flash-Speicher oder über das Netzwerk auf anderen Computern.

Wichtig ist, dass diese Sicherungskopien an einem anderen Ort verwahrt werden als der Rechner. Denn sonst bleibt die Gefahr, dass bei Einbruch, Feuer- oder Wasserschaden doch alle Daten verloren gehen; im Geschäftsleben könnte das den Ruin bedeuten.

[17] Manche Anwenderprogramme führen solche Sicherungen im Abstand einiger Minuten automatisch durch; es kann aber nicht schaden, selbst regelmäßig die Funktion *Sichern* (save) aufzurufen, die von den meisten Anwendungen angeboten wird.

Zum Schluss ein paar Bemerkungen zu den Bezeichnungen. Die Namen Haupt- bzw. Primärspeicher, Sekundärspeicher und Tertiärspeicher weisen auf eine hierarchische Ordnung hin: Was im Primärspeicher keinen Platz findet, kommt in den Sekundärspeicher; was dort nicht hineinpasst, steht im Tertiärspeicher. Es gibt noch eine zweite Beziehung: Der Sekundärspeicher enthält eine (möglicherweise aktuellere) Kopie von Teilen der Information des Tertiärspeichers. Ebenso enthält der Hauptspeicher aktuellere Kopien von Sekundärspeicherinformationen. Jede Schicht in Abbildung 1.4 fungiert also als Cache für die Schicht unter ihr.

Bezeichnungen

Cache-Modell

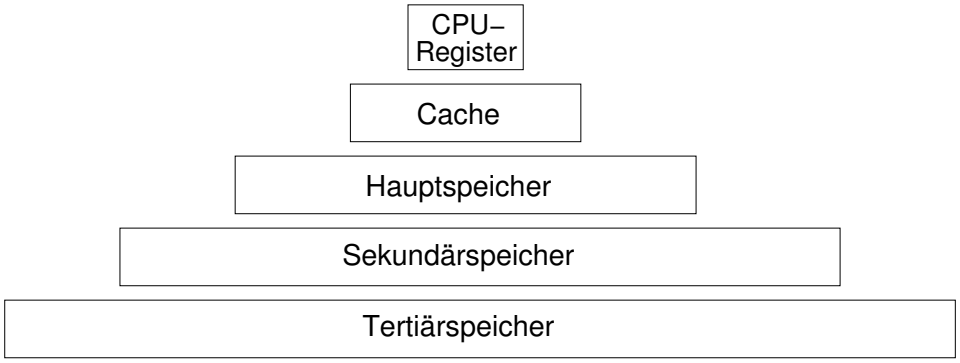

Abbildung 1.4: Jede Schicht arbeitet als Cache für die nächsttiefere Schicht. Die Zugriffszeiten nehmen nach oben hin ab, die Kosten nehmen zu.

Übungsaufgabe 1.6 Sortieren Sie alle bis jetzt behandelten Speicherformen nach Zugriffszeit.

`https://e.feu.de/1801-speicherformen-zugriffszeit`

Sekundär- und Tertiärspeicher werden manchmal auch *Externspeicher* genannt, weil sie in eigenen Gehäusen untergebracht sein können. Man bezeichnet diese Komponenten auch als *Ein-/Ausgabegeräte*, zusammen mit Drucker, Monitor, Tastatur und Maus, denn sie ermöglichen den Fluss von Information in das Rechnerinnere herein und hinaus.

Externspeicher

Ein-/Ausgabegeräte

1.3 Gerätesteuerung

In den vorangegangenen Abschnitten haben wir uns mit wichtigen Komponenten vertraut gemacht, aus denen die Hardware eines Computers besteht: Prozessor, Cache, Hauptspeicher, Sekundär- und Tertiärspeicher. Nun wollen wir sehen, wie die Komponenten eines Computers zusammenwirken, welche Hilfe die Rechnerhardware dabei leistet und welche Aufgaben dabei das Betriebssystem übernimmt.

Eine wichtige Aufgabe des Betriebssystems ist die Kommunikation mit Geräten.

1.3.1 Controller

Bus

Protokoll

In Abschnitt 1.2.1 wurde beschrieben, wie CPU und Hauptspeicher über Adress- und Datenbus miteinander kommunizieren. Auch die Ein-/Ausgabegeräte sind mit Prozessor und Hauptspeicher über einen *Bus* verbunden. Ein Bus ist – grob gesagt – ein Bündel von Leitungen zusammen mit einem *Protokoll*, das genau festlegt, welche Nachrichten über den Bus geschickt werden können und wie diese Nachrichten durch Signale auf den Leitungen dargestellt werden. Ein Bus verbindet – anders als ein Kabel – nicht nur zwei Geräte miteinander, sondern viele Geräte, die alle über den Bus miteinander kommunizieren können. Zu jedem Zeitpunkt kann aber nur *eine* Nachricht über den Bus verschickt werden; welches Gerät den Bus zuerst „ergreift“, ist an der Reihe. Abbildung 1.5 zeigt als Beispiel einen Teil eines Computersystems.[18]

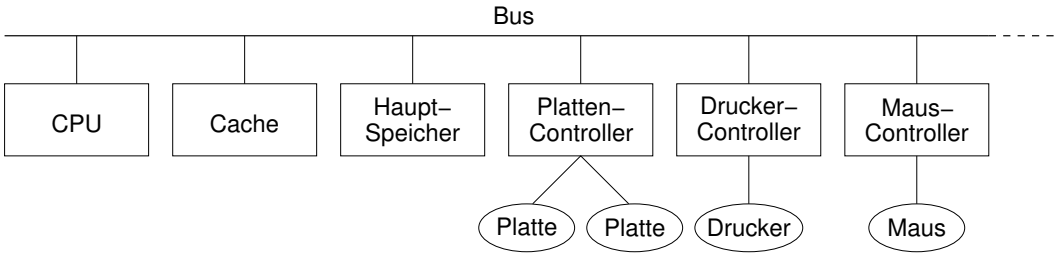

Abbildung 1.5: Ein Computersystem.

Controller

Es fällt auf, dass die Geräte nicht direkt an den Bus angeschlossen sind, sondern über einen sogenannten *Controller*. Das hat folgenden Grund: Wenn die CPU alle Geräte selbst steuern müsste – man denke nur an das ständige Berechnen von Prüfsummen beim Lesen einer Platte –, wäre sie damit so belastet, dass für die eigentliche Programmausführung zu wenig Zeit bliebe. Aus diesem Grund gibt es für jeden Gerätetyp einen Controller, der die Steuerung des „nackten“ Geräts übernimmt. Oft können mehrere Geräte gleichen Typs an einen Controller angeschlossen werden. Die Controller und die CPU arbeiten parallel, aber nur einer von ihnen kann zu einem gegebenen Zeitpunkt den Bus benutzen.

Platten-Controller

Ein Controller ist ein Stück elektronische Hardware. Wie kompliziert er ist, hängt vom jeweiligen Gerätetyp ab. So kommt zum Beispiel eine Maus mit einem sehr einfachen Controller aus, während der Controller einer Magnetplatte schon recht kompliziert ist. Meist wird hierfür ein vollwertiger Prozessor verwendet, der auf einer Platine im Gehäuse des Plattenlaufwerks untergebracht ist. Der Plattencontroller steuert zum Beispiel die Armbewegung, führt Fehlererkennung durch und kann einen schadhaften Sektor der Platte dem Betriebssystem melden. Wenn keine Fehler vorliegen, extrahiert der Controller die Nutzinformation aus dem gelesenen Sektor und schreibt den Block in einen controllereigenen Pufferspeicher.[19]

[18] Darstellungen wie diese sind schematisch; die Architektur realer Rechner kann erheblich komplizierter sein. Zum Beispiel kann es einen speziellen Bus geben, der CPU, Cache und Hauptspeicher miteinander verbindet.

[19] Das geschieht deshalb, weil der für die Übertragung in den Hauptspeicher benötigte Bus

1.3.2 Gerätetreiber

Das Betriebssystem hat die Aufgabe, die Geräte zu steuern; dazu kommuniziert es mit den Controllern. Weil sich die Controller sehr voneinander unterscheiden, ist für jeden Controller ein eigener Teil des Betriebssystems zuständig: ein *Gerätetreiber*; siehe Abbildung 1.6. Als Teil des Betriebssystems ist der Gerätetreiber ein Stück Software.

Gerätetreiber

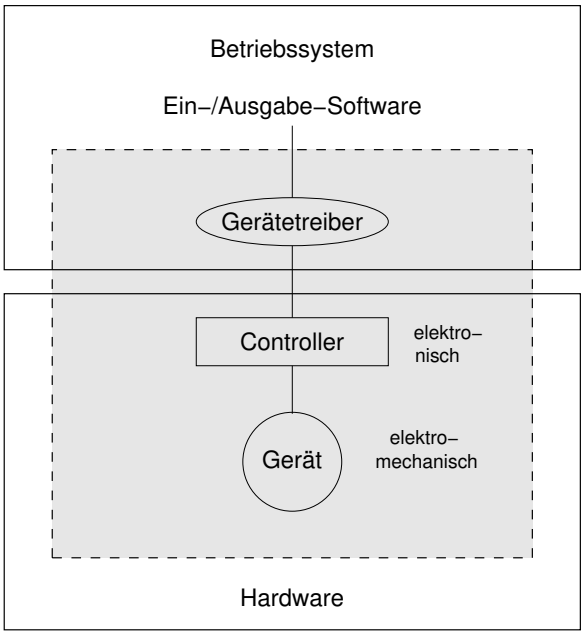

Abbildung 1.6: Ein virtuelles Gerät (grau hinterlegt).

Der Gerätetreiber kommuniziert nur mit dem Controller, nicht direkt mit dem Gerät. Zu diesem Zweck besitzt der Controller mehrere Register, darunter die folgenden:

Register des Controllers

- *Datenausgangsregister* (data-out); hierhin schreibt der Treiber Daten, die für den Controller bestimmt sind;[20]
- *Dateneingangsregister* (data-in), in das der Controller Daten schreibt, die für den Gerätetreiber bestimmt sind;
- *Statusregister* (status); hier kann der Treiber den Zustand des Geräts abfragen, ob es zum Beispiel noch beschäftigt ist oder ob Daten aus dem Dateneingangsregister abgeholt werden können;
- *Kontrollregister* (control); hier hinterlegt der Treiber Befehle an den Controller, zum Beispiel einen Lesebefehl.

Für die Kommunikation mit dem Controller hat jedes seiner Register eine *Portnummer* als Adresse, mit der die CPU das Register ansprechen kann. Die CPU verwendet spezielle Ein-/Ausgabebefehle wie z. B.

gerade besetzt sein könnte; Näheres hierzu finden Sie in Abschnitt 1.3.5.

[20] Hier sind *in* und *out* aus der Sicht des Treibers gemeint, entsprechend Ein- und Ausgaben.

- Lese-Befehl `IN REG, PORT`: liest das Register des Controllers mit der Nummer `Port` und speichert den Inhalt ins CPU-Register `REG`,
- Schreib-Befehl `OUT PORT, REG`: schreibt den Inhalt von CPU-Register `REG` ins Register des Controllers mit der Nummer `PORT`.

Mit diesen Befehlen kann die CPU über besondere Busadressen auf die Controllerregister zugreifen, die am Bus an die durch PORT spezifizierten Adressen angeschlossen sind.

Eine andere Technik für die Kommunikation zwischen der CPU und Controllern ist die *speicherabgebildete Ein-/Ausgabe* (memory-mapped I/O). Dabei werden die Register des Controllers als Teil des Hauptspeichers adressiert, Der auf der CPU laufende Gerätetreiber kann dann die sehr viel schnelleren Prozessorbefehle für den Datenaustausch mit den Controllerregistern verwenden, vergleiche das Beispiel in Abschnitt 1.2.1

memory-mapped I/O

Die Anwendungsprogramme und andere Teile des Betriebssystems können nur über den Gerätetreiber auf das Gerät zugreifen.

Wie sich Controller und Gerätetreiber die Arbeit teilen, ist von Fall zu Fall verschieden. Generell ist die Implementierung von Funktionen in Hardware effizienter als eine Softwarelösung, und sie dient der Abstraktion, weil die Details in der Hardware verborgen werden. Eine Implementierung in Software macht dagegen oft weniger Arbeit und lässt sich leichter ändern.

Hardware vs. Software

Gerätetreiber, Controller und das Gerät bilden konzeptuell eine Einheit, die als *virtuelles Gerät*[21] bezeichnet wird; siehe Abbildung 1.6.

virtuelles Gerät

1.3.3 Exkurs: Abstraktion, Kapselung und Schichtenmodell

Im Modell des virtuellen Geräts werden drei wichtige Konzepte verwendet:

- Abstraktion,
- Kapselung, auch Geheimnisprinzip genannt, und
- Schichtenmodell.

Diese Konzepte spielen in der Informatik eine große Rolle, nicht nur bei den Betriebssystemen, sondern auch beim Software Engineering und beim Algorithmenentwurf. Grund genug also, sich etwas näher mit ihnen zu beschäftigen. Als Beispiel wählen wir ein Magnetplattenlaufwerk, dessen Funktionsweise in Abschnitt 1.2.3 diskutiert worden ist.

Abstraktion

Abstraktion bedeutet, von technischen Details abzusehen und sich auf das Wesentliche, Prinzipielle zu konzentrieren. Wesentlich an der Festplatte ist, dass wir sie als

$$\text{array}[0\,..\,n-1] \text{ of block}$$

[21] *Virtuell* kommt von dem lateinischen Wort *virtus* für Kraft oder Stärke und bedeutet *der Kraft oder Möglichkeit nach vorhanden.*

auffassen können und dass man auf die Blöcke wahlfrei zugreifen kann. Unwesentlich ist dagegen, wie ein Block auf der Platte durch Sektoren realisiert wird und welche Verwaltungsinformation darin gespeichert wird; denn diese zusätzliche Information wird nur intern vom Controller benötigt, aber nicht außerhalb des virtuellen Geräts.

Unwesentlich ist auch, wie weit der Arm des Laufwerks bewegt werden muss, um einen bestimmten Block zu lesen; auf der Abstraktionsebene des virtuellen Geräts – das heißt an der Schnittstelle des Gerätetreibers zum Rest des Betriebssystems – gibt es nur Befehle wie

read(b), write(b), seek(i),

wobei b einen Block bezeichnet und i eine Blocknummer.

Rekapitulieren wir: Das Konzept der Abstraktion besagt, dass man auf höherer Ebene gewisse Details nicht zu kennen *braucht.* Das Prinzip der *Kapselung* geht noch einen Schritt weiter: man *darf* die Details nicht einmal kennen! Aus dieser Forderung ist die alternative Bezeichnung *Geheimnisprinzip* entstanden.

Kapselung

Im Beispiel mit der Magnetplatte bedeutet das: Die Befehle für die Bewegung des Arms existieren nur im Innern des virtuellen Geräts; außerhalb kann man diese Befehle dagegen nicht aufrufen.

Warum ist solch eine strenge Kapselung sinnvoll? Zwei gute Gründe lassen sich anführen. Erstens: Wenn Sie den Auftrag bekommen, einen Gerätetreiber zu schreiben, so haben Sie volle und alleinige Kontrolle darüber, dass Befehle zur Armsteuerung nur auf sinnvolle Weise verwendet werden, also zum Beispiel nicht bei stehendem Motor. Könnte dagegen jeder Benutzer diese Befehle aufrufen, so ergäbe sich hier ein Problem. Und zweitens: Nehmen wir an, das Plattenlaufwerk wird irgendwann durch ein leistungsfähigeres ersetzt. Dann ändern sich möglicherweise auch die Befehle zur Armsteuerung. Würden nun diese Befehle im gesamten Betriebssystem und sogar von Anwenderprogrammen benutzt, so müsste die gesamte Software erneuert werden – ein enorm hoher Aufwand! Bei guter Kapselung braucht man dagegen nur den Gerätetreiber zu ersetzen.[22]

Gründe für Kapselung

Missbrauch verhindern

Änderungen erleichtern

Bei Magnetplatten sind noch weitere Funktionen im virtuellen Gerät gekapselt. Ein Beispiel ist die Erkennung schadhafter Sektoren. Manche Controller unterhalten schon ab Werk ein Verzeichnis aller Sektoren auf den Plattenoberflächen, die nicht einwandfrei funktionieren. Die vom Betriebssystem an den Treiber übergebenen Blöcke werden natürlich nur auf intakte Sektoren geschrieben. Folglich kann es Unterschiede zwischen Block- und Sektornummern geben, die intern vom virtuellen Gerät verwaltet werden.

schadhafte Sektoren

Auch die Optimierung der Armbewegungen, wie wir sie in Übungsaufgabe 1.5 diskutiert hatten, lässt sich im virtuellen Gerät kapseln.

disk scheduling

Soviel zu den Begriffen Abstraktion und Kapselung; sie werden Ihnen im weiteren Studium wiederbegegnen, insbesondere im Zusammenhang mit *ab-*

Objekte und Methoden

[22] Treiber für die wichtigsten Betriebssysteme werden oft vom Gerätehersteller mitgeliefert.

strakten Datentypen, siehe z. B. die Kurse über *Datenstrukturen* und *objektorientierte Programmierung*. In beiden Fällen geht es darum, Objekte und die Methoden (d.h. Operationen), mit denen auf die Objekte zugegriffen werden kann, zu Einheiten zusammenzufassen und den Zugriff auf die Objekte nur über diese Methoden zu gestatten.

Schichtenmodell

Als drittes wesentliches Konzept erkennen wir am virtuellen Gerät den *Aufbau in Schichten;* siehe Abbildung 1.6. In unserem Beispiel ist die oberste Schicht in Software implementiert, die mittlere in elektronischer Hardware und die unterste in mechanischer Hardware.

Die Kommunikation mit der Außenwelt erfolgt nur in der obersten Schicht. In jeder Schicht werden von oben ankommende Aufträge bearbeitet und zur weiteren Bearbeitung an die nächsttiefere Schicht weitergeleitet. Wenn in der untersten Schicht der Auftrag vollständig erledigt ist, wird dort eine Antwort generiert und durch alle Schichten nach oben geleitet. Die oberste Schicht schickt dann die Antwort an den externen Auftraggeber.

Vor- und Nachteile

Ein solcher Ansatz ist natürlich nur dann sinnvoll, wenn jede Schicht einen wesentlichen Beitrag zur Bearbeitung der Aufträge und Antworten leisten kann.[23] Das Schichtenmodell bietet einen großen Vorteil: Die Implementierung einer einzelnen Schicht ist verhältnismäßig einfach, weil nur die beiden Schnittstellen zur nächsthöheren und zur nächsttieferen Schicht realisiert werden müssen und weil im Innern der Schicht nur eine klar umrissenen Teilaufgabe gelöst werden muss.

Beim Entwurf eines Systems nach dem Schichtenmodell muss man eine Abwägung (trade-off) vornehmen: Je „dünner“, also je einfacher man die Schichten macht, desto mehr Schichten werden benötigt und desto höher wird der Verwaltungsaufwand (overhead) für die Kommunikation zwischen den Schichten. Je „dicker“, also je komplexer man die Schichten macht, desto weniger Schichten werden benötigt und desto geringer wird der Verwaltungsaufwand für die Kommunikation zwischen den Schichten. Allerdings werden Änderungen in einer Schicht komplexer und potenziell fehleranfälliger.

1.3.4 Unterbrechungen

Für Menschen ist es ziemlich störend, ständig bei der Arbeit unterbrochen zu werden. Für den Computer sind Unterbrechungen (interrupts) eine natürliche Form der Kommunikation. Warum das so ist, wollen wir jetzt diskutieren.

1.3.4.1 Hardware-Unterbrechungen

Beispiel: Lesezugriff

Nehmen wir an, der Gerätetreiber einer Magnetplatte bekommt den Auftrag, einen bestimmten Block zu lesen. Wie in Abschnitt 1.3.2 besprochen, hinterlegt der Treiber einen entsprechenden Lesebefehl im Kontrollregister des Controllers.

Der Controller kann jetzt mit seiner Arbeit beginnen. Aber wie erfährt die CPU davon, wenn der Leseauftrag ausgeführt ist und der gewünschte Block

[23] „Stempeln und Weiterleiten“ kommt auch bei Behörden allmählich aus der Mode.

im internen Puffer des Controllers bereitliegt? Auf keinen Fall soll die schnelle CPU auf das langsame Gerät warten müssen. Drei bessere Möglichkeiten bieten sich an:

polling

- Die CPU kann – neben ihrer anderen Arbeit – immer wieder das Statusregister des Controllers abfragen, um festzustellen, ob der Auftrag schon erledigt ist; diesen *Abfragebetrieb* nennt man im Englischen *polling*. Liegen die Register im Hauptspeicherbereich – wie im Fall von speicherabgebildeter Ein-/Ausgabe – so lässt sich eine solche Abfrage zwar recht schnell erledigen, aber wenn sie immer wieder erfolglos bleibt, wird insgesamt viel CPU-Zeit damit verbraucht.

interrupt

- Der Controller benachrichtigt die CPU, sobald er den Auftrag ausgeführt hat; hierzu unterbricht er die CPU bei ihrer augenblicklichen Arbeit. Dieser *Unterbrechungsbetrieb* bildet die Grundlage für die Arbeitsweise moderner Computersysteme.

Interrupt-Controller

- Ein modernes Computersystem besitzt einen *Interrupt-Controller*. Der Interrupt-Controller trennt die CPU von verschiedenen Geräten und vermittelt der CPU die Unterbrechungswünsche der dazugehörigen Geräte-Controller.

  Der Interrupt-Controller ist deshalb sowohl mit den verschiedenen Geräte-Controllern verbunden als auch über mehrere Leitungen mit der CPU: Eine Leitung ist mit dem *Unterbrechungseingang* der CPU verbunden, eine Leitung dient zur *Unterbrechungsbestätigung* durch die CPU und eine Leitung für das Übertragen der sogenannten *Unterbrechungsnummer*, die den betreffenden Geräte-Controller, der den Unterbrechungswunsch signalisiert hat, identifiziert.

  Sobald der Geräte-Controller mit einem Auftrag fertig ist, z. B. Lesen eines Datenblocks von der Festplatte, sendet er ein Signal an den Interrupt-Controller. Der Interrupt-Controller teilt diesen Unterbrechungswunsch der CPU mit. Wenn die CPU diese Unterbrechung bearbeiten will, bestätigt sie dem Interrupt-Controller dies über die Bestätigungsleitung. Anschließend überträgt der Interrupt-Controller die Unterbrechungsnummer des Geräte-Controllers an die CPU.

Wir werden uns deshalb in diesem Abschnitt damit befassen, wie der Unterbrechungsbetrieb organisiert ist.

Eine Unterbrechung signalisiert ein bestimmtes Ereignis, das von einer Quelle (Hardware oder Software) gemeldet wird und das von der CPU auf eine spezielle Weise behandelt werden soll. Ein Beispiel für ein solches Ereignis ist das Vorliegen von neuen Daten im Tastaturpuffer. Deswegen gibt es für jeden Typ solcher Ereignisse eine Unterbrechungsnummer und eine dazugehörige *Unterbrechungsroutine* (*Interrupt Service Routine, ISR*), die bei Auftreten eines Ereignisses dieses Typs vom Betriebssystem ausgeführt werden soll.

Unterbrechungsroutine

Das Vorliegen einer Unterbrechung wird der CPU durch das Anlegen eines Signals an ihrem *Unterbrechungseingang* angezeigt. Die daraufhin ablaufende *Unterbrechungsbehandlung* schauen wir uns nun genauer an:

1. Die CPU prüft im Instruktionszyklus zu Beginn der Holphase, ob ein Unterbrechungssignal anliegt. Falls nicht, dann wird der Instruktionszyklus weiter ausgeführt und der nächste Befehl des aktuell rechnenden Prozesses ausgeführt.

2. Falls ein Unterbrechungssignal anliegt, wird automatisch (durch entsprechende Schaltungen in der CPU) der aktuelle Registersatz in den systemeigenen Speicher (z. B. ein *Stack*[24] im Cache auf dem CPU-Chip) kopiert. Mit dieser Kopie kann später der Zustand vor der Unterbrechung wiederhergestellt werden.

3. Die CPU schaltet nun in den Systemmodus[25] und muss die passende Unterbrechungsroutine aufrufen. Eine einfache Möglichkeit, dies zu bewerkstelligen, ist es, die Startadresse einer allgemeinen Unterbrechungsroutine, die an einer bekannten Adresse im Speicher steht, in das Befehlszählregister der CPU zu laden. Dies bewirkt einen Sprung an diese Adresse, und die allgemeine Unterbrechungsroutine wird ausgeführt.

4. Die allgemeine Unterbrechungsroutine muss zuerst die Unterbrechungsnummer feststellen (z. B. durch Polling der Geräte oder durch Abfrage beim Interrupt-Controller). Im *Unterbrechungsvektor*, siehe Abbildung 1.7, steht an dieser Stelle die Startadresse der passenden Unterbrechungsroutine. Diese Adresse wird in das Befehlszählregister der CPU geladen und damit in diese Routine gesprungen.

   Unterbrechungsvektor

5. Die Instruktionen in der Unterbrechungsroutine haben die Aufgabe, den Unterbrechungswunsch des Geräts, der durch das Unterbrechungssignal angezeigt wird, zu bearbeiten: Sie kann zum Beispiel den Gerätetreiber darüber informieren, dass der Controller seinen Lesebefehl ausgeführt hat und die Daten nunmehr bereitstehen.

6. Am Ende der Unterbrechungsroutine muss der Registersatz des unterbrochenen Prozesses, wie er vor der Unterbrechung vorlag, durch den Befehl *Return-from-Interrupt* wiederherstellt werden. Glücklicherweise hat die CPU zu Beginn der Unterbrechungsbehandlung diesen Registersatz im systemeigenen Speicher abgelegt, siehe Schritt 2. Von dort wird nun der Registersatz etc. wiederhergestellt. Die unterbrochene Berechnung des Prozesses wird nun weiter ausgeführt als ob es nie eine Unterbrechung gegeben hätte.

   Return-from-Interrupt

[24] Ein Stack (Stapel) ist eine Datenstruktur. Ein Element darf nur ganz von oben auf den Stapel eingefügt oder aus dem Stapel entfernt werden.

[25] Die CPU besitzt mindestens zwei Modi wie z. B. System- und Benutzermodus. Nur im Systemmodus dürfen privilegierte Befehle ausgeführt werden.

Eine Unterbrechung führt deshalb immer zum Aufruf der passenden Unterbrechungsroutine im Systemmodus, d. h. im Kernel des Betriebssystems. Daher können Unterbrechungen als Mechanismus gesehen werden, mit dem man das Betriebssystem aufrufen bzw. aktivieren kann.

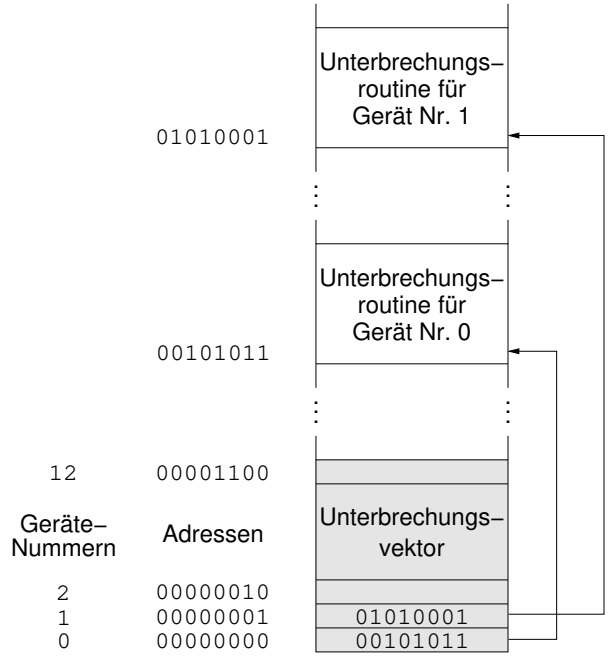

Abbildung 1.7: Ein Unterbrechungsvektor.

Mehrere Unterbrechungen

Es kann vorkommen, dass während der Bearbeitung einer Unterbrechung ein weiterer Unterbrechungswunsch eintrifft. Zur Vermeidung eines Durcheinanders sind verschiedene Maßnahmen möglich.

Unterbrechungen sperren

Zum einen lässt sich bei vielen Prozessoren der Unterbrechungseingang vorübergehend außer Betrieb setzen (interrupt disabled), so dass weitere Unterbrechungssignale wirkungslos bleiben.[26] So kann die CPU die zuerst eingetroffene Unterbrechung ungestört bearbeiten. Dieses Verfahren bietet sich auch in anderen Situationen an, wenn die CPU bei der Ausführung kritischer Abschnitte eines Programms nicht unterbrochen werden darf; hierauf werden wir in Abschnitt 2.3 zurückkommen.

Prioritäten vergeben

Zum anderen kann man jedem Unterbrechungswunsch eine Priorität zuordnen; Unterbrechungen niederer Priorität können dann von solchen mit höherer Priorität unterbrochen werden.

Übungsaufgabe 1.7 Nach jeder Ausführung eines Befehls überprüft die CPU, ob an ihrem Unterbrechungseingang ein Signal angekommen ist. Schleicht sich hier durch die Hintertür der ineffiziente Abfragebetrieb ein, den wir durch Einführung des Unterbrechungsbetriebs eigentlich hatten vermeiden wollen?

`https://e.feu.de/1801-unterbrechungen-abfragebetrieb`

[26] Man spricht in diesem Zusammenhang von *maskierten* Unterbrechungen.

Übungsaufgabe 1.8 Was passiert, wenn während der Bearbeitung einer niedrig priorisierten Unterbrechung das Signal einer höher priorisierten Unterbrechung beim Interrupt-Controller eintrifft?

`https://e.feu.de/1801-sperrung-unterbrechungseingang`

Wir haben oben gesehen, dass Geräte (genauer: ihre Controller) Unterbrechungen der CPU auslösen, wenn sie ihren Auftrag ausgeführt haben oder wenn ein Fehler aufgetreten ist.

Hardware-Unterbrechung

Wir bezeichnen diese Unterbrechung, die durch ein externes Gerät verursacht wird, als *Hardware-Unterbrechung*, die nicht von dem gerade ausgeführten Programm verursacht wird. Diese Unterbrechung kann nicht wieder reproduziert werden, wenn das Programm noch einmal ausgeführt wird, da das Programm nichts mit der Unterbrechung zu tun hat.

1.3.4.2 Software-Unterbrechung

Software-Unterbrechungen

Aber auch die Software kann Unterbrechungen (sogenannte *traps*) auslösen. Der Unterschied zu der Hardware-Unterbrechung ist, dass eine Software-Unterbrechung von dem geraden ausgeführten Programm verursacht wird. So eine Unterbrechung ist reproduzierbar: wenn das Programm noch einmal ausgeführt wird, passiert die Unterbrechung genau an derselben Stelle des Programms.

Für eine Software-Unterbrechung kann es ganz verschiedene Ursachen geben, zum Beispiel will das Programm eine Division durch Null oder einen Zugriff auf nichtexistierende oder geschützte Hauptspeicheradressen ausführen; für solche Unterbrechungen wird auch die Bezeichnung *Ausnahme* (exception) verwendet.

Ausnahmen

Systemaufrufe

Eine andere Art von Software-Unterbrechung stellt dagegen keine Ausnahme dar; sie ist vielmehr ein übliches Verfahren zur Kontrollübergabe an das Betriebssystem: Wann immer ein Programm Dienste des Betriebssystems in Anspruch nehmen will, führt es einen *Systemaufruf* (system call) durch. Wir werden in Abschnitt 1.6 ausführlicher darauf eingehen, welche Systemaufrufe ein Betriebssystem üblicherweise anbietet. Die verfügbaren Systemaufrufe bilden zusammen die *Programmierschnittstelle* zwischen den Anwenderprogrammen und dem Betriebssystem; siehe auch Abschnitt 1.1.

Parameterübergabe

Solch ein Systemaufruf hat die Form eines Funktionsaufrufs. Insbesondere können dabei auch Parameter übergeben werden, entweder direkt in einem Register der CPU oder indirekt durch Angabe der Anfangsadresse des zu übergebenden Datenbereichs im Hauptspeicher; dieses Vorgehen empfiehlt sich bei großen Datenmengen, wie zum Beispiel Bildschirminhalten. Man kann zur Parameterübergabe auch den Stapel des Prozesses benutzen.

trap

Wenn der Systemaufruf seine Parameter für die Übergabe vorbereitet hat, führt er eine besondere Instruktion[27] *trap* aus, die die eigentliche Unterbrechung

[27] Die Begriffe *Befehl, Anweisung* und *Instruktion* werden in diesem Kurs synonym verwendet.

auslöst. Diese Instruktion ist bei allen Systemaufrufen dieselbe; der gewünschte Systemdienst wird beim Aufruf durch einen Parameter bezeichnet.

Jetzt geschieht dasselbe wie bei einer Hardware-Unterbrechung: Befehlszähler- und Registerinhalte werden gerettet, und in Abhängigkeit vom übergebenen Parameter für den gewünschten Dienst springt die allgemeine Unterbrechungsroutine in die entsprechende gewünschte spezielle Unterbrechungsroutine des Betriebssystems. Bei Systemaufrufen werden meistens Daten an das aufrufende Programm zurückgegeben. Die Unterbrechung ist hier also der Mechanismus für den Aufruf einer Betriebsystemfunktion.

Aktivierung vom Betriebssystem

Wer in einer Hochsprache programmiert, wird möglicherweise nicht immer bemerken, dass sein Programm Systemaufrufe auslöst: Zum Beispiel gibt es in Pascal einen Standardbefehl namens *read*, mit dem ein Datensatz einer Datei gelesen werden kann. Ein solcher Befehl wird vom Compiler automatisch in einen entsprechenden Systemaufruf übersetzt, ohne dass der Programmierer die Details kennen muss.

1.3.5 Direkter Speicherzugriff (DMA)

Wir hatten in Abschnitt 1.3.4.1 besprochen, was geschieht, wenn ein Plattenlaufwerk einen Leseauftrag ausgeführt hat: Der gewünschte Datenblock steht im Pufferspeicher des Controllers bereit, und der Controller schickt ein Unterbrechungssignal an die CPU, um sie hierüber zu informieren.[28] Wie gelangt der Block nun in den Hauptspeicher? Hier gibt es zwei unterschiedliche Verfahren.

1. Bei der *unterbrechungsgesteuerten Ein-/Ausgabe* schreibt der Controller jeweils ein Wort in sein Dateneingangsregister und löst eine Unterbrechung aus; siehe auch Abschnitt 1.3. Dadurch wird der Gerätetreiber gestartet; er veranlasst, dass das Wort von der CPU in Empfang genommen und in den Hauptspeicher geschrieben wird. Wenn ein Block 512 Bytes enthält und ein Wort zwei Bytes umfasst, wird die CPU bei diesem Verfahren 256-mal unterbrochen, bevor der Datenblock endlich im Hauptspeicher steht!

   unterbrechungsgesteuerte Ein-/Ausgabe

2. Eigentlich ist die CPU für solche niederen Übertragungsdienste zu schade. Deshalb wird für schnelle Peripheriegeräte häufig ein anderes Verfahren gewählt: der *direkte Speicherzugriff* (direct memory access = DMA). Hierbei wird im Computer ein spezieller DMA-Controller eingesetzt, der selbständig über den Bus Daten in den Hauptspeicher übertragen kann, ohne die CPU zu bemühen. Auch der Gerätecontroller muss für das DMA-Verfahren ausgelegt sein.

   direkter Speicherzugriff
   DMA

Der Gerätetreiber teilt dem Gerätecontroller über dessen Register die Nummer des zu lesenden Blocks mit und dem DMA-Controller die Anfangsadresse des Hauptspeicherbereichs, in den die Daten übertragen werden sollen. Danach kann sich die CPU anderen Aufgaben widmen. Der Gerätecontroller setzt sich

Arbeitsweise von DMA

[28] Wir nehmen in diesem Abschnitt an, dass im Unterbrechungsbetrieb gearbeitet wird, d. h. das Betriebssystem Unterbrechungen und die zuvor beschriebene Unterbrechungsverarbeitung unterstützt.

nun über spezielle Leitungen mit dem DMA-Controller in Verbindung. Wann immer das nächste Wort im Register des Gerätecontrollers bereitsteht, sendet er über eine *Anforderungsleitung* ein Signal an den DMA-Controller; dieser schreibt die entsprechende Hauptspeicheradresse auf den Adressbus und signalisiert über die *Bestätigungsleitung* dem Gerätecontroller, die Übertragung durchzuführen. Dann zählt der DMA-Controller die Adresse für das nächste Wort hoch. Erst wenn alle Wörter des gesamten Blocks übertragen sind, löst er eine Unterbrechung der CPU aus. Ein Beispiel für den Aufbau eines DMA-fähigen Systems zeigt Abbildung 1.8.

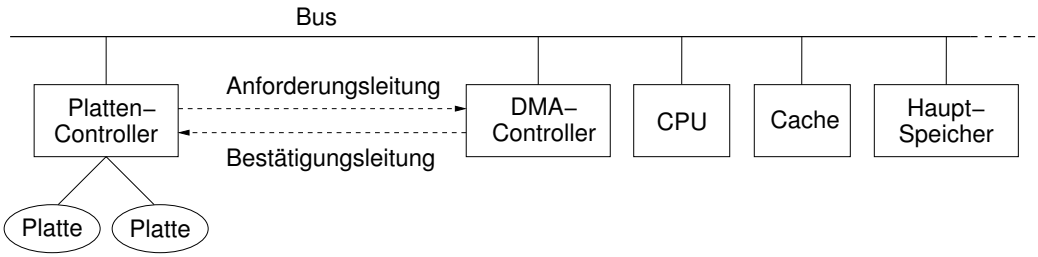

Abbildung 1.8: Eine mögliche Rechnerarchitektur für direkten Speicherzugriff (DMA).

Übungsaufgabe 1.9 Das Ziel des DMA-Verfahrens liegt in einer Entlastung der CPU. Wieso kann es trotzdem vorkommen, dass die Arbeit der CPU verzögert wird, während DMA-Vorgänge ablaufen?

`https://e.feu.de/1801-dma`

handshaking

Eine Kommunikation wie die zwischen DMA- und Gerätecontroller wird im Englischen als *handshaking* bezeichnet; sie stellt einen einfachen Fall eines Protokolls dar.

1.3.6 System- und Benutzermodus; Speicherschutz

Start des Rechners

Ur-Lader

Auch das Betriebssystem selbst ist ein Programm. Wenn der Rechner eingeschaltet wird, steht ein kleiner Teil des Systems – der sogenannte *Ur-Lader* (bootloader) – in einem speziellen Teil des Hauptspeichers, der seinen Inhalt auch nach Abschalten des Stroms behält. Meist ist der Ur-Lader in einem ROM (read only memory) fest eingebrannt, das auch Firmware genannt wird bzw. BIOS bei PCs.

Der Ur-Lader wird beim Einschalten des Rechners automatisch ausgeführt und lädt zunächst den eigentlichen Lader von der Magnetplatte.[29] Der Lader wird nun gestartet und lädt das Betriebssystem – zumindest seine wesentlichen Teile – in den Hauptspeicher. Dieser Vorgang wird als *hochfahren* (booting) bezeichnet.

booting

[29] Hierdurch wird erreicht, dass der Lader aktualisiert werden kann, wenn später einmal eine erweiterte Betriebssystemversion installiert wird.

Wenn jetzt ein Anwenderprogramm ausgeführt wird, könnte es im Prinzip versuchen, Teile des Hauptspeicherbereichs zu überschreiben, in dem das Betriebssystem steht; dazu braucht das Programm ja nur die entsprechenden Speicherzellen zu adressieren, wie wir in Abschnitt 1.2.1 besprochen haben. Ebenso könnte ein Benutzerprogramm sich selbst oder andere Programme, die sich gerade im Hauptspeicher befinden, verändern.

Schutz von Speicher und Geräten

Ob eine solche Veränderung nun versehentlich oder mit Absicht geschieht: die Folgen können schlimm sein.[30] Aus diesem Grund kann es einem Benutzerprogramm nicht gestattet werden, auf beliebige Teile des Hauptspeichers zuzugreifen. Auch der Sekundärspeicher muss vor unbefugtem Zugriff geschützt werden, denn von dort wird ja das Betriebssystem beim nächsten Einschalten geladen, und es befinden sich dort auch andere Programme. Schließlich bedarf auch der Zugriff auf die anderen Geräte der Kontrolle; wenn zum Beispiel mehrere Programme gleichzeitig auf den Drucker zugreifen, kann sonst ein Durcheinander entstehen.

System- und Benutzermodus

Um diesen Schutz zu gewährleisten, können moderne Prozessoren im *Systemmodus*[31] (system mode) oder im *Benutzermodus* (user mode) arbeiten;[32] zur Unterscheidung wird ein besonderes Bit im Prozessor verwendet. Alle Anwendungsprogramme laufen im Benutzermodus. Bestimmte *privilegierte Maschinenbefehle* können aber nur im Systemmodus ausgeführt werden. Damit solch ein Schutzmechanismus wirksam ist, muss natürlich der Befehl zum Umschalten vom Benutzer- in den Systemmodus selbst auch privilegiert sein. Muss man also schon privilegiert sein, um Privilegien zu bekommen?

privilegierte Befehle

Dank des Betriebssystems nicht! Dieses Problem wird so gelöst: Wenn eine Unterbrechung auftritt – insbesondere wenn ein Benutzerprogramm einen Systemaufruf auslöst – schaltet die privilegierte Prozedur zur Unterbrechungsbehandlung den Prozessor in den Systemmodus und startet dann die entsprechende Unterbrechungsroutine; vergleiche Abschnitt 1.3.4. Sie ist Teil des Betriebssystems und deshalb vertrauenswürdig. Bevor die Unterbrechungsroutine terminiert, schaltet sie in den Benutzermodus zurück.

Kein Benutzer kann also direkt auf ein Gerät zugreifen; man muss zu diesem Zweck das mit höheren Rechten ausgestattete Betriebssystem um Hilfe bitten.

Übungsaufgabe 1.10 Beim *Return-from-Interrupt* stellt die CPU den Registersatz vor der Unterbrechung wieder her. Was gehört dazu?

`https://e.feu.de/1801-return-from-interrupt`

[30] Der erste Autor erinnert sich an eigene Erfahrungen mit einem kleinen Rechner, der in Maschinensprache programmiert wurde. Wenn man sich bei den Sprungadressen im Hexadezimalcode verrechnet hatte, traten die merkwürdigsten Fehler auf.

[31] Der Systemmodus wird auch Supervisor-Modus genannt.

[32] Beim Intel 8088 war diese Möglichkeit nicht vorhanden. Deshalb kannte MS-DOS keine Schutzmechanismen, was unter anderem die Anfälligkeit gegen Würmer und Viren erklärt.

Übungsaufgabe 1.11 Beim *Return-from-Interrupt* wird der Modus gesetzt auf...

https://e.feu.de/1801-moduswechsel

Mit diesen Hilfsmitteln lässt sich auch ein wirksamer Speicherschutz realisieren; und zwar auf folgende Weise:

Adressraum

Bevor ein Programm gestartet wird, weist ihm das Betriebssystem einen bestimmten Bereich im Hauptspeicher zu, seinen sogenannten *Adressraum*. Das Programm darf nur auf solche Adressen zugreifen, die in seinem eigenen Adressraum liegen. Das bedeutet: Alle möglicherweise benötigten Daten, aber auch das Programm selbst müssen in diesem Adressraum enthalten sein.

Basis- und Grenzregister

Ein zusammenhängender Adressraum lässt sich durch die Inhalte von zwei speziellen Registern der CPU festlegen. Im *Basisregister* (base register) steht die niedrigste Adresse des Adressraums; das *Grenzregister* (limit register) enthält die Länge des Adressraums, also die Differenz aus der höchsten und der niedrigsten erlaubten Adresse. Die Inhalte von Basis- und Grenzregister können nur mit speziellen Maschinenbefehlen verändert werden. Diese Befehle sind privilegiert.

Wann immer das Programm zur Laufzeit versucht, auf eine Speicherzelle zuzugreifen, wird zunächst überprüft, ob die angegebene Adresse im erlaubten Bereich liegt; siehe Abbildung 1.9. Wenn das der Fall ist, wird der Speicherzugriff durchgeführt, andernfalls wird eine Software-Unterbrechung ausgelöst, wie in Abschnitt 1.3.4 beschrieben.

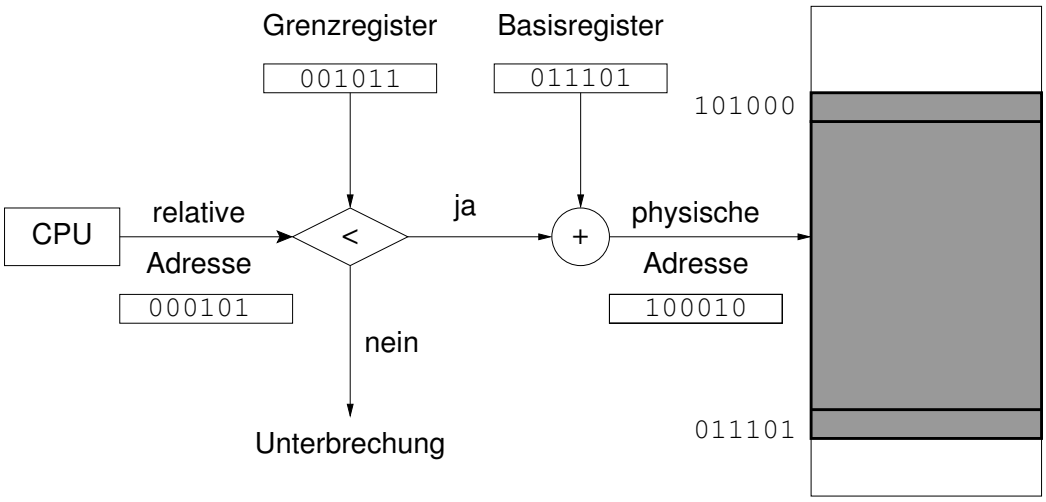

Abbildung 1.9: Speicherschutz mit Basis- und Grenzregister.

Mit einer leichten Modifikation dieser Technik lässt sich gleich noch ein zweites Problem erledigen. Wenn ein Programm vom *Compiler* in Maschinensprache übersetzt wird, steht noch nicht fest, wo im Hauptspeicher der Adressraum des Programms später liegen wird – das hängt ja auch davon ab, welche

anderen Programme dann gerade im Hauptspeicher stehen. Der Compiler kann deshalb an die Befehle und die Daten zunächst nur *relative Adressen* vergeben wie etwa „17 Wörter hinter dem Anfang dieses Programmstücks“.[33]

relative Adressen

Wird das Programm nun in seinen Adressraum geladen, stimmen die im Programm verwendeten relativen Adressen nicht mit den absoluten Adressen überein, an denen die Befehle und Daten tatsächlich stehen. Man könnte deshalb vor dem Laden alle relativen Adressen im Programm um die Anfangsadresse des Adressraums erhöhen. Diese Lösung ist nicht so elegant; sie benötigt nämlich zusätzliche Information darüber, wo im Programm die Adressen stehen.[34] Diese Hilfsinformation müsste eigens vom Computer generiert werden. Viel günstiger ist es, die Neuberechnung der Adressen nach der Formel

absolute Adressen

$$\text{absolute Adresse} = (\text{Basisregister}) + \text{relative Adresse}$$

erst zur Laufzeit durchzuführen! Das heißt: Wenn das Programm eine Speicherzelle adressieren will, wird die im Programm enthaltene relative Adresse zur niedrigsten Adresse des Adressraums addiert; diese steht im Basisregister.

Damit wird auch sichergestellt, dass die absolute Adresse nicht unterhalb des Adressraums liegt. Vor der Addition ist noch der erste Test aus Abbildung 1.9 nötig, damit der Zugriff nicht über die obere Grenze des Adressraums ausgeht.

Weil die Programme zusammen mit ihren Adressräumen beliebig im Hauptspeicher verschoben werden können, spricht man bei diesem Verfahren von *relokierbaren Programmen* (relocatable code).

relokierbare Programme

Übungsaufgabe 1.12 Warum darf ein Anwendungsprozess nicht auf den Unterbrechungsvektor zugreifen?

`https://e.feu.de/1801-unterbrechungsvektor`

Übungsaufgabe 1.13 Muss der Befehl, mit dem der Unterbrechungseingang der CPU außer Betrieb gesetzt (maskiert) wird, privilegiert sein?

`https://e.feu.de/1801-maskieren-privilegiert`

[33] Auch die relativen Adressen werden in Binärdarstellung angegeben. Würde der Adressraum bei 0 beginnen, könnte man sie als absolute Adressen verwenden. Die Adresse 0 gehört aber oft zum vom Betriebssystem genutzten Bereich (Kernel Space).

[34] Erinnern wir uns an Abschnitt 1.2.1: Den Worten allein sieht man nicht an, ob sie Befehle, Daten oder Adresse darstellen.

1.4 Prozesse

In Abschnitt 1.3 haben wir untersucht, wie das Betriebssystem mit Hilfe des Mechanismus der Unterbrechung die Geräte steuert. Wir wollen uns in diesem Abschnitt damit beschäftigen, wie das Betriebssystem die Ausführung von Programmen ermöglicht und warum wir das Gefühl haben, dass mehrere Prozesse *quasiparallel* laufen.

1.4.1 Prozesszustände und Übergänge

In Abschnitt 1.3.4 haben wir beobachtet, was geschieht, wenn die CPU bei der Ausführung eines Programms unterbrochen wird: Die Adresse des nächsten auszuführenden Befehls und die Registerinhalte werden im Systemstapel gespeichert; damit wird die CPU für andere Aufgaben frei. Später können die Registerinhalte wieder geladen werden, und die CPU kann ihre Arbeit an der richtigen Stelle fortsetzen, so als hätte es die Unterbrechung nicht gegeben.

Prozess

Ein Programm, das sich gerade in Ausführung befindet, heißt *Prozess*. Zum Prozess gehört aber nicht nur ein Programm (Code, der ausgeführt werden soll), sondern auch der *Prozesskontext* bestehend aus

Prozesskontext

- den Registerinhalten, insbesondere Befehlszähler und
- Grenzen des Adressraums, sowie der
- Prozessnummer,
- Priorität (benötigt)
- Modus (im Systemmodus- oder im Benutzermodus) und
- Zustand

PCB

und anderen Informationen. Diese Angaben werden im *Prozesskontrollblock* (process control block = PCB) zusammengefasst. Der PCB steht im Speicherbereich des Betriebssystems und ist somit vor dem Zugriff durch Anwendungsprogramme geschützt.

Ein Prozess wird zu irgendeinem Zeitpunkt *erzeugt*, z. B. durch einen anderen Prozess, und zu einem späteren Zeitpunkt *beendet*, wenn er mit der Bearbeitung fertig ist. Dazwischen kann er mehrfach zwischen drei Zuständen wechseln, die in Abbildung 1.10 dargestellt sind.[35]

Prozesszustände

Nach der Erzeugung eines Prozesses wird der PCB vom Betriebssystem angelegt und initialisiert, d. h. die benötigten Ressourcen wie z. B. ein Speicherbereich werden zugeteilt. Nun ist der Prozess bereit für die Bearbeitung und er geht in den Zustand *bereit*.

[35] Solche Diagramme werden gern zur Beschreibung von Systemen verwendet, die endlich viele innere Zustände (dargestellt durch Ovale) annehmen können. Die Beschriftung an einem Pfeil gibt an, durch welchen äußeren Reiz der betreffende Zustandswechsel ausgelöst wird. Dort könnte auch vermerkt sein, wie das System dabei nach außen reagiert. Diese Systeme heißen *endliche Automaten*. Sie werden in Kurs *Grundlagen der Theoretischen Informatik* behandelt.

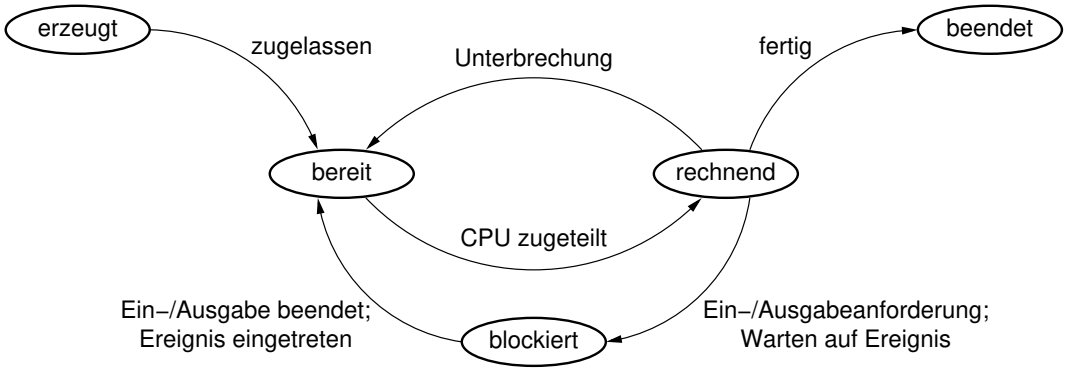

Abbildung 1.10: Mögliche Prozesszustände

Bei den Einprozessorsystemen, wie wir sie in diesem Kurs betrachten, ist zu jedem Zeitpunkt genau ein Prozess im Zustand *rechnend*; nämlich der Prozess, dessen Programm gerade von der CPU ausgeführt wird. Jeder andere existierende Prozess ist entweder *bereit* und bewirbt sich mit den übrigen bereiten Prozessen um die Zuteilung der CPU, oder er ist *blockiert* und wartet darauf, dass seine Ein-/Ausgabeanforderung erledigt wird oder ein bestimmtes Ereignis eintritt.

rechnend

bereit

blockiert

Wenn zum Beispiel ein laufendes Benutzerprogramm auf eine Eingabe von der Tastatur wartet, führt es einen Systemaufruf durch. Bei der Durchführung des Systemaufrufs durch das Betriebssystem wird der Prozess vom Zustand *rechnend* in den Zustand *blockiert* versetzt, da er ohne die Eingabedaten nicht weiterarbeiten kann.

Beispiel: Ein-/Ausgabe

Danach ist die CPU frei, um andere Prozesse zu bearbeiten. Wenn dann die Daten bereitstehen und der Tastaturcontroller eine Unterbrechung auslöst, wird der Benutzerprozess in den Zustand *bereit* gebracht, denn seine Arbeit kann nun weitergehen. Das heißt aber nicht, dass er sofort die CPU zugeteilt bekommt: Welcher der bereiten Prozesse als nächster rechnen darf, entscheidet der *CPU-Scheduler*.[36] Der Scheduler definiert die Ausführungsreihenfolge der bereiten Prozesse im nächsten Zeitabschnitt. Er kann seine Entscheidungen nach unterschiedlichen Strategien treffen, siehe auch Abschnitt 1.4.2. Der Dispatcher führt den Prozesswechsel durch.

CPU-Scheduler

Scheduling-Strategien

Ein rechnender Prozess kann auch durch einen Systemaufruf mitteilen, dass er auf ein bestimmtes *Ereignis* warten will; er wird dann *blockiert* und kommt in eine spezielle Warteschlange, in der möglicherweise noch andere blockierte Prozesse stehen, die auf dasselbe Ereignis warten. Wenn dann dieses Ereignis eintritt, kann der Prozess, welcher es auslöst, ein *Signal* abschicken. Dieses Signal bewirkt, dass alle Prozesse aus der Warteschlange wieder in den Zustand *bereit* versetzt werden.

Signal

Wenn ein Prozess im Zustand *rechnend* ist, muss er irgendwann die CPU wieder abgeben. Grundsätzlich gibt es zwei Arten, wie entschieden wird, wie ein Prozess vom Zustand *rechnend* in den Zustand *bereit* gehen kann:

[36] Das entsprechende deutsche Wort *Planer* ist unüblich.

nicht präemptiv

präemptiv

1. Bei einem *nicht präemptiven* System gibt ein Prozess die CPU freiwillig ab. Man bezeichnet so ein System auch als *kooperatives System*.
2. Bei einem *präemptiven* System wird einem Prozess die CPU entzogen. Die meisten modernen Systeme sind präemptive Systeme.

Übungsaufgabe 1.14 Mehrere Prozesse sollen in einem Rechensystem ausgeführt werden. Welche Aussagen sind richtig?

`https://e.feu.de/1801-scheduling`

1.4.2 Scheduling-Strategien für nicht präemptive Systeme

Wir betrachten zwei Scheduling-Strategien für ein nicht präemptives System:

FCFS

- Bei FCFS (first-come, first-served) darf zuerst rechnen, wer zuerst kommt; vergleiche Abschnitt 1.2.3. Dieses Verfahren ist leicht zu implementieren; es genügt, eine *Warteschlange* (queue) einzurichten, bei der immer der erste Prozess als nächster an die Reihe kommt und neu eintreffende bereite Prozesse sich hinten anstellen. Der Nachteil: Ein früh eintreffender Prozess mit hohem Rechenzeitbedarf oder einer Endlosschleife hält alle späteren Prozesse auf.

SJF

- Das Verfahren *SJF* (shortest job first) kann den Nachteil von FCFS vermeiden; hier werden die Prozesse im Zustand *bereit* in der Reihenfolge aufsteigenden Rechenzeitbedarfs bearbeitet: die kurzen zuerst.[37] Voraussetzung für eine sinnvolle Anwendung von SJF ist, dass sich die benötigte Rechenzeit (bis zur nächsten Unterbrechung) aus Erfahrungswerten gut vorhersagen lässt.[38] Unter dieser Voraussetzung ist SJF sehr effizient, wie die folgende Aufgabe zeigt.

Übungsaufgabe 1.15 Gegeben sei eine feste Menge von endlich vielen Prozessen mit bekannten Rechenzeiten. Beweisen Sie, dass SJF die Gesamtwartezeit minimiert, also die Summe aller Wartezeiten der einzelnen Prozesse. Dabei ist die Wartezeit eines Prozesses diejenige Zeit, in der sich der Prozess im Zustand *bereit* befindet.

`https://e.feu.de/1801-sjf-wartezeit`

[37] Man beachte die Ähnlichkeit zwischen SJF und dem Verfahren SSTF in Abschnitt 1.2.3. Beides sind *gierige* (greedy) Strategien, die stets den im Moment maximal möglichen Vorteil suchen.

[38] Allgemeine Vorhersagen über die Rechenzeit eines beliebigen Programms zu treffen, ist nicht möglich. Selbst die Frage, ob ein Programm überhaupt jemals anhält, ist *unentscheidbar*, wie in der Theoretischen Informatik bewiesen wird.

In einem dynamischen System kommen allerdings immer wieder neue Prozesse hinzu. Je mehr kurze Prozesse generiert werden, desto länger müssen die langen Prozesse warten. Wenn also immer neue kurze Prozesse erzeugt werden, kommen die langen Prozesse *nie* an die Reihe. Dieses Problem wird *Verhungern* (engl. *starvation*) genannt.

Verhungern

Übungsaufgabe 1.16 Wie kann man die SJF-Strategie so modifizieren, dass das Problem des Verhungerns vermieden wird?

https://e.feu.de/1801-sjf-probleme

Das Verfahren SJF eignet sich besonders gut für den *Stapel-* oder *Batch-Betrieb*, bei dem der Rechner gleich einen ganzen Schub von Aufträgen (jobs) erhält, die keine Interaktion mit dem Benutzer erfordern und regelmäßig auszuführen sind, so dass man ihre Laufzeiten in etwa kennt.

Batch-Betrieb

1.4.3 Scheduling-Strategien für präemptive Systeme

Moderne Rechner werden meist im *Time-Sharing-Betrieb* verwendet: Mehrere Benutzer können zur selben Zeit an einem Rechner arbeiten, und jeder kann gleichzeitig mehrere Programme laufen lassen, zum Beispiel in einem Fenster einen Compiler, in einem anderen ein Werkzeug für elektronische Post und in einem dritten Fenster einen WWW-Browser.[39]

Time-Sharing-Betrieb

Diese „Gleichzeitigkeit“ ist natürlich eine Illusion, denn bei einem Computer mit nur einer CPU kann nur ein einziger Prozess rechnend sein. Der Eindruck von Gleichzeitigkeit entsteht dadurch, dass der Scheduler in schnellem Wechsel jedem bereiten Prozess ein gewisses Quantum an Rechenzeit zukommen lässt. Wenn das Quantum verbraucht wird, wird dem Prozess die CPU entzogen. Danach erfolgt die Umschaltung zwischen den Prozessen, die so schnell ist, dass die Unterbrechungen nicht spürbar werden.

Drei Strategien können wir uns für präemptive Systeme vorstellen:

1. Ein sehr einfaches Verfahren namens *Round Robin* ist weit verbreitet, bei dem jeder Prozess im Zustand *bereit* vom Scheduler eine *Zeitscheibe* (time slice) *derselben Länge* an Rechenzeit zugewiesen bekommt und die Prozesse im Zustand *bereit* reihum bedient werden. Dazu kann man sie in der Reihenfolge ihres Eintreffens in einer kreisförmigen Warteschlange speichern. Bei der Strategie Round Robin werden alle Prozesse gleich behandelt. Diese Eigenschaft kann auch ein Nachteil für die interaktiven Prozesse sein, die oft eine Eingabe oder Ausgabe benötigen, wenn z. B. die Länge der Zeitscheibe zu groß gewählt wird.

   Round Robin
   Zeitscheibe

2. Eine Verbesserung von Round Robin ist, dass der Scheduler einem Prozess die Länge der Zeitscheibe in Abhängigkeit von der *Priorität* des

[39] Statt von Time-Sharing spricht man manchmal auch von *Multitasking*.

Prozesses oder davon, wieviel Rechenzeit der Prozess insgesamt schon verbraucht hat, zuweist.[40] Der Scheduler wählt dann den Prozess als nächsten aus, der die höchste Priorität hat. Dieses Vorgehen kann aber dazu führen, dass bei ständig neu erzeugten Prozessen mit höherer Priorität die niedrig priorisierten Prozesse, trotz längerer Zeitscheibe, nie rechnend werden.

3. Die Priorität eines Prozesses kann sogar nach seinem Verhalten dynamisch vergeben werden. Beispielsweise warten die interaktiven Prozesse oft auf eine Ein-/Ausgabe und sie benötigen für deren Verarbeitung nur eine kurze Zeitscheibe. Insbesondere benötigen sie hierfür aber oft die CPU, um auf die Ein-/Ausgabe reagieren zu können. Um diesem Verhalten zu entsprechen, können sie vom Scheduler anhand dieser Charakteristik erkannt und eine kurze Zeitscheibe sowie eine höhere Priorität bekommen.

Zeitgeber

Die technische Realisierung mit der Zeitscheibe kann folgendermaßen geschiehen: Eine Hardware (*Zeitgeber* (Timer)) wacht darüber, dass der Prozess sein Quantum nicht überschreitet; oft ist dafür ein eigener Chip im Rechner vorhanden, der an den Takt der CPU angeschlossen ist. Die Länge der zugewiesenen Zeitscheibe wird in einem Register des Zeitgebers gespeichert. Nach jeder verstrichenen Zeiteinheit wird der Inhalt dieses Registers um Eins verringert. Ist der Wert bei Null angekommen, so ist die zugewiesene Zeitscheibe abgelaufen, und der Zeitgeber löst eine Unterbrechung der CPU aus. Der Prozess wird unterbrochen und wieder in die Menge der bereiten Prozesse eingereiht; vergleiche Abbildung 1.10. Nun kommt ein anderer Prozess an die Reihe.

Übungsaufgabe 1.17 Muss der Zugriff auf das Register des Zeitgebers privilegiert sein?

`https://e.feu.de/1801-timer`

Wie effizient dieses Verfahren arbeitet, hängt von der Länge der Zeitscheiben ab. Die nächste Aufgabe zeigt, dass extreme Werte unzweckmäßig sind.

Dispatcher

Das Umschalten zwischen Prozessen bezeichnet man auch als *Kontextwechsel* (context switch). Während der Scheduler die Strategie für die Rechenzeitvergabe festlegt, führt der *Dispatcher* die eigentlichen Kontextwechsel durch. Da solche Kontextwechsel häufig vorkommen, ist es wichtig, dass der Dispatcher möglichst schnell arbeitet.

Übungsaufgabe 1.18 Welche Gefahren drohen, wenn beim Verfahren Round Robin die Zeitscheiben zu lang oder zu kurz gewählt werden?

`https://e.feu.de/1801-zeitscheiben`

[40] Diese Angabe steht ebenfalls im Prozesskontrollblock.

1.5 Rekapitulation: ein Gerätezugriff

In diesem Abschnitt wollen wir im Zusammenhang darstellen, was das Betriebssystem leistet, wenn es bei der Ausführung eines Programms auf eine Ein-/Ausgabeanforderung stößt, etwa auf einen Lesebefehl

read(f,b),

bei dem der Datensatz b von der Datei f gelesen werden soll. Die meisten Teilschritte haben wir in den vorangegangenen Abschnitten bereits diskutiert.

So wurde am Ende von Abschnitt 1.3.4 festgestellt, dass schon bei der Übersetzung des Programms der Hochsprachen-Befehl *read* durch einen entsprechenden Systemaufruf ersetzt wird, der den Ein-/Ausgabeteil des Betriebssystems startet. Außerdem muss die *logische* Beschreibung der gewünschten Daten – „Datensatz b in Datei f“ – in eine *physische* Beschreibung abgebildet werden, wie etwa „Block Nr. i auf der Magnetplatte M“. Dies geschieht zur Laufzeit des Programms mit Hilfe des *Dateisystems* (file system), auf das wir in Abschnitt 2.4.2 ausführlicher eingehen werden.

Dateisystem

Übungsaufgabe 1.19 Warum wird nicht schon beim Compilieren die logische Datenbeschreibung (logische Adresse) durch die physische Beschreibung (physische Adresse) ersetzt?

`https://e.feu.de/1801-adressumrechnung`

Im Time-Sharing-Betrieb kann es vorkommen, dass viele Prozesse kurz nacheinander auf ein Gerät zugreifen wollen. Deshalb wird zu jedem Speichergerät eine eigene *Geräte-Warteschlange* (device queue) eingerichtet, an die der Ein-/Ausgabeteil des Betriebssystems Aufträge anhängen kann; jeder Auftrag wird mit der Nummer des Prozesses versehen, der ihn erteilt hat. Der Gerätetreiber holt die Aufträge einzeln aus der Warteschlange ab und führt sie dem Controller zu. Dabei ist der Treiber nicht an die Reihenfolge in der Warteschlange gebunden; er kann vielmehr versuchen, durch geschickte Wahl des nächsten Auftrags die Zugriffszeit des Geräts zu minimieren, wie in Übungsaufgabe 1.5 besprochen.[41]

Geräte-Warteschlange

Nun verfolgen wir, was bei Ausführung des zu *read* gehörigen Systemaufrufs in einem Prozess P geschieht:

1. Eine Software-Unterbrechung wird ausgelöst; der Kontext von P wird zunächst gerettet, dann wird der geräteunabhängige Ein-/Ausgabeteil des Betriebssystems aufgerufen.

[41] Beim Drucker muss man anders vorgehen. Wenn nämlich mehrere Prozesse ihre Ausgabe zeilen- oder absatzweise zum Drucker schicken und sich dabei gegenseitig unterbrechen, entsteht auf dem Papier ein Durcheinander. Deshalb werden die Ausgaben zunächst nach Prozessen getrennt in einer *Spooler-Datei* gesammelt, die erst dann gedruckt wird, wenn der Prozess seine Druckausgabe abgeschlossen hat.

2. Dieser prüft zunächst, ob die Parameter zulässig sind und ob die benötigten Daten schon im Cache im Hauptspeicher stehen; in diesem Fall werden sie in den Adressraum von P kopiert, und nach der Rückkehr von der Software-Unterbrechung kann der Prozess P weiter rechnen.

3. Stehen die Daten nicht im Cache, ist ein Plattenzugriff erforderlich. Der Prozess P wird deshalb in den Zustand *blockiert* versetzt. Der Leseauftrag wird an die Geräte-Warteschlange des Plattenlaufwerks angehängt und der Gerätetreiber benachrichtigt.

4. Sobald der Gerätetreiberprozess rechnend wird, entnimmt er den Auftrag der Geräte-Warteschlange. Der Gerätetreiber reserviert im betriebssystemeigenen Bereich Speicherplatz für die zu lesenden Daten, bestimmt die gerätespezifische Datenblockadresse etc. und überträgt einen Lesebefehl an den Gerätecontroller. Der Treiber kann nun andere Aufgaben ausführen, er wird später informiert, wenn der Lesebefehl ausgeführt worden ist, siehe Schritt 6.

5. Der Gerätecontroller bestimmt zuerst die physische Adresse, dann führt er den Leseauftrag aus und überträgt zusammen mit dem DMA-Controller die Daten in den reservierten Bereich im systemeigenen Speicher; währenddessen können auf der CPU beliebige andere Prozesse rechnen. Danach wird eine DMA-Unterbrechung ausgelöst.

6. Bei der Unterbrechungsverarbeitung wird mittels des Unterbrechungsvektors die DMA-Unterbrechungsroutine gestartet. Sie informiert den Gerätetreiber, dass der Lesebefehl ausgeführt worden ist.

7. Sobald der Gerätetreiber diese Information empfängt, sieht er in der Gerätewarteschlange nach, von welchem Prozess der Leseauftrag zu dieser Information stammt, und informiert den geräteunabhängigen Ein-/Ausgabeteil des Betriebssystems.

8. Sobald der geräteunabhängige Ein-/Ausgabeteil des Betriebssystems die Information empfängt, kopiert er die Daten aus dem systemeigenen Speicher in den Adressraum des Prozesses P und versetzt P in den Zustand bereit. Der Systemaufruf ist damit erfolgreich bearbeitet worden.

9. Sobald der Prozess P wieder rechnen darf setzt er seine Arbeit hinter dem *read* fort.

An manchen Stellen mag man sich fragen, warum der Ablauf gerade so und nicht anders organisiert ist. In der Regel sprechen gute – wenn auch nicht immer zwingende – Gründe für die hier beschriebene Organisation. Beispielsweise werden die gelesenen Daten erst im systemeigenen Speicherbereich abgelegt, weil der Platz für die Daten im Adressraum von P erst in Schritt 8 zugewiesen werden soll oder zwischendurch schon wieder ausgelagert worden sein könnte, vergleiche dazu Abschnitt 2.1.

1.6 Programmierschnittstelle

In den vorangegangenen Abschnitten haben wir bei der Rechnerhardware begonnen und uns von unten nach oben (bottom-up) in die Aufgaben und Strukturen eines Betriebssystems eingearbeitet. Nun können wir einen Blick auf ein ganzes System werfen.

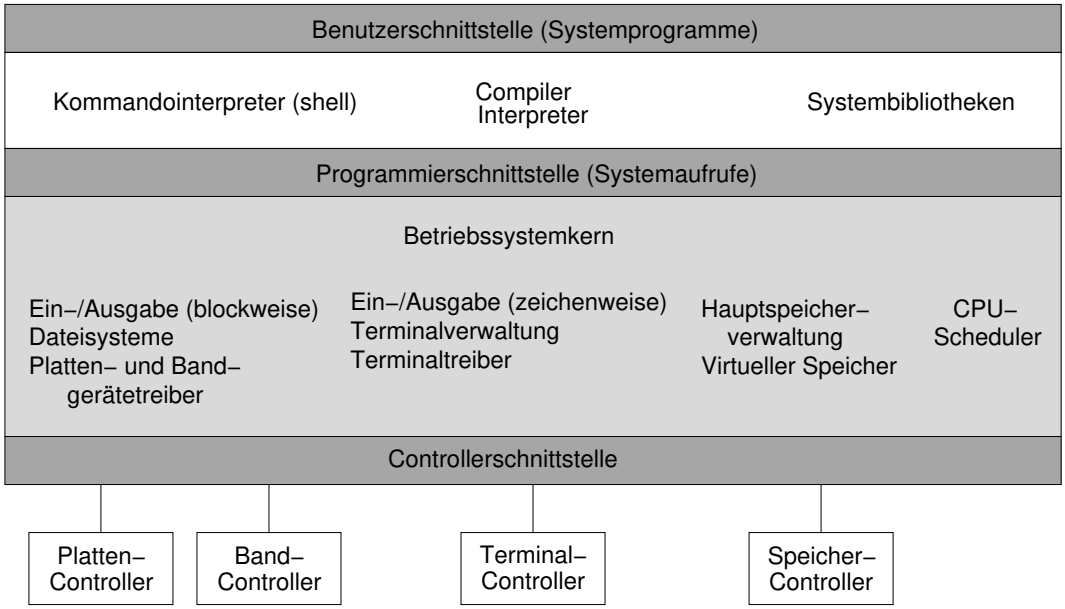

Abbildung 1.11: Die Struktur des Betriebssystems UNIX.

In Abbildung 1.11 ist die Struktur von UNIX skizziert. Der sogenannte *Kern* (kernel) des Betriebssystems ist hellgrau dargestellt. Er enthält geräteunabhängige Teile für blockweise Ein- und Ausgabe, wie sie bei Platten- und Bandgeräten gebräuchlich ist, und für zeichenweise Ein- und Ausgabe wie bei Modems und Terminals. Ein weiterer Teil des Kerns ist für die Hauptspeicherverwaltung zuständig. Der CPU-Scheduler gehört zum Kern, und in UNIX und Linux auch die Gerätetreiber. Daneben gibt es weitere Teile für die Erledigung anderer Aufgaben.

Kern

Wir sehen in Abbildung 1.11, dass der Kern zwei Schnittstellen aufweist: Die Verbindung nach unten zur Hardware erfolgt über die *Controllerschnittstelle*. Die *Programmierschnittstelle* stellt die Dienste des Betriebssystems nach oben, also für die Programme zur Verfügung; dies geschieht durch die Gesamtheit der *Systemaufrufe* (system calls), wie wir sie in Abschnitt 1.3.4 besprochen haben.

Schnittstellen

Oberhalb der Programmierschnittstelle sind in Abbildung 1.11 die *Systemprogramme* eingezeichnet. Sie bilden zusammen die *Benutzerschnittstelle*, mit der wir uns in Abschnitt 1.8 befassen werden.

In diesem Abschnitt wollen wir eine Übersicht über einige wichtige Dienste geben, die als Systemaufrufe angeboten werden.[42] Diese Systemaufrufe lassen

Systemaufrufe

[42]Dabei legen wir uns nicht auf ein spezielles Betriebssystem fest; in UNIX und Linux

sich grob folgenden Aufgabenbereichen zuordnen:

- Prozesse,
- Dateien,
- Information und
- Kommunikation.

Prozesse

Einige Beispiele für Systemaufrufe im Zusammenhang mit *Prozessen* haben wir bereits kennengelernt: Ein Prozess kann *auf ein Ereignis warten* (wait event) und wird blockiert, bis ein anderer Prozess durch ein *Signal* mitteilt, dass das erwartete Ereignis eingetreten ist (signal event); vergleiche das Ende von Abschnitt 1.4.1.

anhalten und abbrechen

Ein Prozess muss freiwillig *anhalten* können (halt), wenn er mit seiner Arbeit fertig ist; er wird dadurch in den Zustand beendet versetzt. Wenn ein Fehler auftritt, muss der Prozess *abgebrochen* werden (abort); dabei sollte es möglich sein, eine Fehlermeldung auszugeben oder den Inhalt von Registern und Adressraum für die Fehlersuche zu speichern (dump).

Ein Prozess sollte ein anderes Programm *laden* (load) und *ausführen* (execute) können. So kann ein neuer Prozess entstehen, der entweder erst beendet sein muss, bevor der Vaterprozess weiterrechnen kann, oder neben seinem Vaterprozess existiert; hierfür ist oft ein besonderer Systemaufruf *Prozess!erzeugen* (create process) zuständig.[43] Manchmal will der Vaterprozess beim Kindprozess bestimmte *Attribute setzen* (set attributes), etwa einen Adressraum im Adressraum des Vaters oder eine maximale Rechenzeit. Es kann auch notwendig sein, den Kindprozess zu *terminieren* (terminate).

Dateien

Den Begriff *Datei* (file) haben wir schon früher verwendet, ohne genau zu sagen, was damit gemeint ist. Eine Datei ist, grob gesagt, eine Sammlung zusammengehöriger Information. Dabei kann es sich um ein Programm in Maschinencode handeln, ein druckbares PostScript-Dokument, eine Videosequenz oder um eine HTML-Seite. Dateien befinden sich oft im Sekundär- und Tertiärspeicher, manchmal aber auch im Hauptspeicher (sogenannte *RAM-Disk)*, um den Zugriff zu beschleunigen. Die Magnetplatte eines PC enthält meist sehr viele Dateien; schon das Betriebssystem und die Systemprogramme belegen oft mehrere hunderttausend Dateien.

Aus der Sicht des Benutzers sind Dateien *abstrakte Datentypen* (siehe auch Kurse 1661 oder 1663 Datenstrukturen), ähnlich wie virtuelle Geräte; vergleiche Abschnitt 1.3.3. Sie sind in *logische Datensätze* (records) aufgeteilt, die einzelne Informationseinheiten darstellen und oft dieselbe Länge haben. So kann zum Beispiel in einer Personaldatei für jede Mitarbeiterin ein Datensatz vorhanden sein. Es gibt Operationen, um Dateien zu erzeugen, zu löschen und um einzelne Datensätze zu lesen oder zu schreiben. Dabei kann der Zugriff auf die Datensätze sequentiell oder wahlfrei sein. Manchmal kann man über einen

sind jedenfalls Aufrufe dieser Art vorhanden.

[43] In UNIX und Linux wird solch ein Prozess mit *fork* erzeugt und mit *exec* ausgeführt.

Schlüssel zugreifen: Bei der Personaldatei gibt man dann die Personalnummer vor und erhält den Datensatz des entsprechenden Mitarbeiters.[44]

Das Dateisystem hat die Aufgabe, diese logisch-abstrakte Sicht auf die physische Organisation einer Datei in Blöcke abzubilden; vergleiche den Anfang von Abschnitt 1.5; das kann oft dadurch geschehen, dass jeweils k Datensätze gleicher Größe in einen Block geschrieben werden.

anlegen und löschen

Man benötigt dazu Systemaufrufe, mit denen sich eine Datei physisch *anlegen* (create) oder *löschen* (delete) lässt. Vor dem ersten Zugriff muss man eine Datei *öffnen* (open), nach dem letzten Zugriff *schließen* (close). Durch das Öffnen wird die Datei in eine Liste der geöffneten Dateien im PCB eingetragen; bei den folgenden Zugriffen braucht dann nicht jedesmal aufs neue die physische Adresse der Datei bestimmt zu werden. Außerdem lässt sich damit verhindern, dass zwei Benutzer sich beim Zugriff auf dieselbe Datei stören.

öffnen und schließen

lesen und schreiben

Es gibt Systemaufrufe zum *Lesen* (read) und zum *Schreiben* (write) eines Blocks. Ferner lässt sich eine Datei vor- oder zurück-„spulen“, um auf einen bestimmten Block direkt zugreifen zu können; die Vorstellung des Umspulens stammt dabei von Magnetbandgeräten, auf denen ursprünglich große Dateien gespeichert wurden.

Information Zeit und Datum

Systemaufrufe zur Beschaffung von *Information* können vom System die *Zeit* erfragen oder das *Datum*. Manche Systeme stellen Aufrufe bereit, welche die Nummer der laufenden Betriebssystemversion oder eine Liste mit den Nummern aller vorhandenen Prozesse liefern.

Kommunikation

Die *Kommunikation* zwischen Prozessen kann auf zwei verschiedene Arten erfolgen: über den Versand von *Nachrichten* (message passing) oder durch einen *gemeinsamen Speicherbereich* (shared memory).[45]

Versand von Nachrichten

Beim *Versand von Nachrichten* gibt es zwei Varianten: Der *verbindungslose* (connectionless) Datenaustausch entspricht dem Versenden eines Briefs: Man gibt die Adresse des Empfängers und des Absenders an und überlässt die Übermittlung dem System. Ein Beispiel für den verbindungslosen Austausch ganz kurzer Nachrichten sind die Signale, die oben schon erwähnt wurden. Für längere Nachrichten gibt es Systemaufrufe zum *Senden* (send) und *Empfangen* (receive), die besonders beim *Client-Server-Betrieb*[46] wichtig sind.

Beim *verbindungsorientierten* (connection-oriented) Datenaustausch wird dagegen erst eine Verbindung zwischen Sender und Empfänger aufgebaut, über die dann die Kommunikation erfolgt; dem entspricht ein Telefongespräch. Hierfür braucht der Sender Systemaufrufe zum *Öffnen* und *Schließen* der Verbindung; der Empfänger kann schon auf die Verbindung *warten* (wait) oder sie doch wenigstens *akzeptieren* (accept). Sobald die Verbindung zustande gekommen ist, kann man Nachrichten *schreiben* und *lesen*. In UNIX und Linux werden solche Verbindungen durch *pipes* realisiert.

[44] Diese datenbankähnliche Zugriffsmethode wird von der Programmiersprache COBOL unterstützt; trotzdem wollen wir hier keine Empfehlung für diese altertümliche Sprache geben. Moderne Programmiersprachen bieten hierzu die Datentypen Dictionary oder hashmap an.

[45] Bei Rechnernetzen kommt nur der Versand von Nachrichten für die Kommunikation in Frage.

[46] Beim Client-Server Betrieb übermitteln die Clients ihre Dienstanfragen an einen Server, der diese dann verarbeitet und die Ergebnisse zurück gibt.

gemeinsamer Speicherbereich

Viel effizienter ist es, für die Kommunikation zwischen Prozessen einen *gemeinsamen Speicherbereich* zu verwenden, in den der Sender schreibt und von dem der Empfänger liest, denn dann kann man die schnellen CPU-Befehle für den Hauptspeicherzugriff benutzen. Hier treten aber zwei Schwierigkeiten auf: Zum einen achtet das Betriebssystem ja darauf, dass verschiedene Prozesse getrennte Adressräume haben; siehe Abschnitt 1.3.6. Hier braucht man Systemaufrufe, um (vorübergehend) einen Teil des einen Adressraums in den anderen abzubilden. Zum anderen kann es Probleme geben, wenn sich die Prozesse beim Lesen und Schreiben stören. Hier sind *Synchronisationsmechanismen* erforderlich, wie wir sie in Abschnitt 2.3 behandeln werden.

1.7 Generierung von Prozessen

Bei der Diskussion der Programmierschnittstelle eines Betriebssystems in Abschnitt 1.6 hatten wir gesehen, dass das Betriebssystem Systemaufrufe zur Erzeugung und zur Beendigung von Prozessen bereitstellen muss. In diesem Abschnitt wollen wir an einem einfachen Beispiel untersuchen, wie diese Systemaufrufe in UNIX[47] eingesetzt werden.

ein Beispiel

Wir wollen für einen Wiedergabeprozess, z. B. einen Audioplayer, eine „Benutzungsoberfläche“ schreiben, die dem Benutzer erlaubt, den Player zu starten und zu stoppen. Aus Gründen der Einfachheit und Systemunabhängigkeit benutzen wir hier einen sehr simplen Player.

Es geht in dem Beispiel[48] also darum, in einem laufenden Prozess namens forkdemo einen Kindprozess beeper zu erzeugen, der eine Folge von Pieptönen produziert, bis er von seinem Elternprozess forkdemo wieder beendet wird. Wir haben die Programme der Prozesse in C geschrieben, weil Linux und UNIX in C und C++ geschrieben sind.

```
/* beeper.c, Endlosschleife mit Pieptönen       */
/* kompilieren: cc -o beeper beeper.c           */
/* starten mit: beeper                          */
/* beenden mit: kill oder Control-C             */

# include<stdio.h>

int main()
{
    while (!0) {
      printf ("%c",07);
    }
}
```

[47] Zur Familie der UNIX-Systeme gehört insbesondere auch Linux. Die aufgeführten Kommandos sind daher auch unter Linux einsetzbar.

[48] Auf der Webseite unter `http://www.fernuni-hagen.de/ks/1801/` steht der Programm-Code bereit zum Kopieren und Ausführen.

Das C-Programm forkdemo für den Elternprozess enthält eine Schleife, in der die Eingabe von 1, 2 oder 0 angefordert wird; Eingabe von 1 startet den Kindprozess, Eingabe von 2 beendet ihn wieder, und wenn man 0 eingibt, werden der Elternprozess und auch der Kindprozess beendet.

```
/* forkdemo.c, erzeugt und beendet Kindprozess beeper */
/* kompilieren: cc -o forkdemo forkdemo.c             */
/* starten mit: forkdemo                              */
/* beeper starten lassen mit 1, stoppen mit 2,        */
/* Programm beenden mit 0                             */

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
int main(int argc, char*argv[])

{
int prozessnr, ergebnis, piep;
                        /* piep = 0, Piepton ist aus */
                        /* piep = 1, Piepton ist an */
char c ;

    piep = 0;
    prozessnr = -1;      /* kein Prozess gestartet */
    do {
         printf("1: Piepton starten\n");
         printf("2: Piepton stoppen\n");
         printf("0: Programm beenden\n");
         scanf("%c",&c);  /* Eingabe von Tastatur einlesen */
         printf("\n");
         printf("Eingegeben ist %c\n",c);

         if (c == '1' && piep == 0) {
           piep = 1;  /* Piepton wird nur einmal gestartet */
           prozessnr = fork(); /* neuen Prozess erzeugen: */
           if (prozessnr == 0) {/* fork liefert prozessnr */
                                /* den Wert 0 im Kindprozess */
                           /* und die Prozessnummer des */
                           /* Kindprozesses im Elternprozess */
             ergebnis = execvp( "beeper", argv);
                           /* 'beeper' starten */
             exit(0);
           }
         }/* if */
         if (c == '2' && prozessnr != -1) {
```

```
            piep = 0;
                      /* Piepton wird abgeschaltet */
            ergebnis = kill( prozessnr,9);
                   /* Kindprozess beenden */
          }/* if */
    } /* do */
    while (c != '0');
    if ( piep != 0) {
      ergebnis = kill(prozessnr,9);  /* nicht vergessen, den */
                                     /* Piepton abzuschalten */
    }/* if */
}/* main */
```

fork

Schauen wir uns die interessanten ersten beiden Fälle näher an. Bei Eingabe von 1 wird ein Kindprozess erzeugt. Hierzu wird der Systemaufruf *fork* benutzt. Die Funktion *fork* hat kein Argument. Ihr Aufruf bewirkt, dass eine genaue Kopie vom Elternprozess erzeugt und bereit gemacht wird; auch die Registerinhalte, insbesondere der Inhalt des Befehlszählers, werden kopiert.

Der Kindprozess konkurriert mit dem Elternprozess um Zuteilung der CPU. Beide führen als nächstes die Anweisung

if prozessnr == 0 **then** ...

aus. Und hier gehen Eltern- und Kindprozess nun getrennte Wege: Denn der vorausgegangene Funktionsaufruf

prozessnr = fork();

liefert im Elternprozess die Prozessnummer des Kindprozesses zurück, im Kindprozess aber den Wert 0!

execvp

Nur der Kindprozess wird also als nächstes die Anweisung

ergebnis = execvp('beeper', argv);

ausführen. Sie bewirkt, dass das gegenwärtige Programm des Kindprozesses, das ja mit dem seines Elternprozesses identisch ist, durch das Programm in der Datei beeper ersetzt wird, und veranlasst die Ausführung dieses Programms vom Programmanfang.[49] Jetzt wird eine endlose Folge von Pieptönen erzeugt.

In dem Fall, dass beim Aufruf von *execvp* im Kindprozess ein Fehler auftritt — es könnte ja sein, dass gar keine Datei namens beeper im aktuellen Verzeichnis existiert — erhält die Variable *ergebnis* einen Wert; danach veranlasst die Exit-Anweisung die Beendigung des Kindprozesses. Ohne diese Vorsichtsmaßnahme würde der Kindprozess jetzt dasselbe Programm ausführen wie sein Elternprozess, was nur Verwirrung stiften könnte.

[49] Die zweite Variable der Funktion *execvp* kann zur Übergabe weiterer Parameter zwischen den beiden Hochkommata verwendet werden; in unserem Beispiel machen wir hiervon keinen Gebrauch.

top

Mit dem Befehl *top* der Benutzerschnittstelle von UNIX kann man sich eine Liste aller existierenden Prozesse und ihres Ressourcenverbrauchs ausgeben lassen, die ständig aktualisiert wird. Hier ist jetzt neben dem Prozess forkdemo auch sein Kind beeper zu sehen.

Beim Elternprozess forkdemo hat die Variable *prozessnr* als Wert die Prozessnummer von beeper; sie ist von null und von der Prozessnummer von forkdemo verschieden, so dass es nicht zu Verwechselungen kommen kann. Der Test in der Anweisung

if prozessnr == 0 **then** ...

verläuft für den Elternprozess also negativ. Folglich durchläuft er die Schleife erneut und fordert eine neue Eingabe an.

kill

Wir gehen einmal davon aus, dass der Benutzer die Pieptöne gerne wieder abstellen möchte und deshalb 2 eingibt. Dann wird der Systemaufruf

ergebnis = kill (prozessnr, 9)

durchgeführt. Er bewirkt, dass das Signal mit der Nummer 9 an den Prozess mit der Nummer *prozessnr* geschickt wird. Dies ist der Kindprozess beeper, und Signal Nummer 9 bewirkt seine Beendigung.[50]

man

Über Varianten der hier benutzten Systemaufrufe und ihre Funktion kann man sich mit Hilfe des Befehls *man* informieren. Geben Sie zum Beispiel einmal

man fork

ein, um nachzulesen, was ein Kind von seinem Elternprozess erbt.

1.8 Benutzungsschnittstelle

In Abschnitt 1.6 haben wir uns mit der Programmierschnittstelle eines Betriebssystems beschäftigt; nun wenden wir uns der *Benutzungsschnittstelle* zu. Sie dient dem Benutzer zusammen mit den Schnittstellen der Anwendungsprogramme zur Kommunikation mit dem Rechner. Wie angenehm es sich mit einem Rechner arbeiten lässt, hängt entscheidend von der Gestaltung der Benutzungsschnittstelle ab. Weit verbreitet sind heute graphische Benutzungsoberflächen mit Fenstern, Menüs und Maussteuerung.

Systemprogramme

Während die Programmierschnittstelle durch die Gesamtheit aller Systemaufrufe gegeben ist, besteht die Benutzungsschnittstelle aus der Gesamtheit aller *Systemprogramme* (system programs). In Abbildung 1.11 sieht man am Beispiel von UNIX, wie die Systemprogramme über die Programmierschnittstelle auf die Systemaufrufe zugreifen.

[50] In der von *top* generierten Prozessliste wird beeper danach möglicherweise noch als *zombie* aufgeführt; der beendete Prozess bekommt aber weder Rechenzeit noch Speicherplatz.

Auch die Systemprogramme lassen sich grob den folgenden Aufgabenbereichen zuordnen:

- Programme,
- Dateien und Verzeichnisse,
- Information und
- Kommunikation.

Sie sehen: Diese Liste entspricht der Einteilung der Systemaufrufe in Abschnitt 1.6. Der Unterschied besteht darin, dass Systemprogramme oft auf einer höheren Abstraktionsebene angesiedelt sind als Systemaufrufe, und dass sie in eine Benutzungsoberfläche integriert sind. Ein Beispiel für eine solche integrierte Oberfläche ist das Common Desktop Environment (CDE)[51]. Es vereinigt viele Systemprogramme mit den nachfolgend beschriebenen Funktionen in sich; man kann auch CDE selbst als ein großes Systemprogramm auffassen.

Programme

editieren, übersetzen, binden, laden

Fehler suchen

Gehen wir die einzelnen Bereiche durch. Von Systemprogrammen werden alle Benutzeraktivitäten unterstützt, die mit dem Herstellen und Ausführen von *Programmen* zu tun haben. Dazu gehört ein *Editor* zum Schreiben der Programmtexte,[52] ein *Compiler* oder *Interpreter* für die verwendete Programmiersprache, ein *Binder* (linker), der einzelne Module zu einem Lademodul zusammenfasst, und schließlich den *Lader* (loader), der das Lademodul in den Hauptspeicher bringt; vergleiche Abschnitt 1.3.6. Außerdem ist ein *Debugger* (bug = Wanze) bei der Fehlersuche hilfreich.

Dateien und Verzeichnisse

Zugriffsrechte setzen

verwalten

Systemprogramme helfen dem Benutzer, Dateien *anzulegen*, zu *kopieren, zu drucken, umzubenennen* und zu *löschen*. Auch das Setzen der *Zugriffsberechtigung*, d. h. die Festsetzung, welcher Benutzer wie auf eine Datei zugreifen darf, kann mit einem Systemprogramm erfolgen.[53] Zusammengehörige Dateien kann man in *Verzeichnissen* (directories) zusammenfassen. Ein Verzeichnis kann selbst andere Verzeichnisse enthalten. So lassen sich hierarchische Strukturen aufbauen, in denen man sich gut zurechtfindet. Für die Verwaltung von Verzeichnissen gibt es ähnliche Systemprogramme wie für Dateien. Wir werden uns hiermit in Abschnitt 2.4 ausführlicher beschäftigen.

Information
Systemdaten

Neben *Zeit* und *Datum* lassen sich wichtige Systemdaten abfragen wie *freier Speicherplatz*, *CPU-Auslastung* und eine Liste der anderen Benutzer, die zur Zeit am Rechner arbeiten.

Kommunikation

Während die Systemaufrufe insbesondere mit der Kommunikation zwi-

[51] oder die freien Linux Oberflächen KDE und GNOME.

[52] Bei Editoren gibt es ein breites Spektrum, vom zeilenorientierten *vi* in UNIX bis hin zu graphischen Editoren mit komfortabler Maussteuerung. Für das Programmieren in einer bestimmten Sprache kann ein *syntaxgesteuerter* Editor zweckmäßig sein, der Verstöße gegen die Regeln der Sprache schon beim Eingeben erkennt, die Programmzeilen sinnvoll einrückt und Schlüsselwörter fett druckt, z. B. *Emacs*.

[53] An diesem Beispiel lässt sich der Unterschied zwischen Systemaufruf und Systemprogramm verdeutlichen: In UNIX und Linux gibt es den Systemaufruf *chmod*, mit dem der Besitzer einer Datei festlegen kann, wer außer ihm – d. h. seine Gruppe oder alle Benutzer – die Datei lesen, beschreiben oder ausführen dürfen. Unter CDE lässt sich im File Manager ein Menü öffnen, das die Zugriffsrechte einer ausgewählten Datei anzeigt und zu ändern erlaubt.

schen Prozessen zu tun haben, sind Systemprogramme auf höherer Ebene für die Kommunikation zwischen Benutzern zuständig; diese können an voneinander weit entfernten Rechnern arbeiten. Besonders bekannt sind *elektronische Post* (electronic mail), *Dateitransfer* (file transfer, z.B. mit dem *file transfer protocol*) und *Benutzerzugriff auf entfernte Rechner* (remote login). Hieran sind auch die Rechnernetze beteiligt, die die Rechner miteinander verbinden. Näheres hierzu finden Sie in den Kurseinheiten 3 und 4 dieses Kurses.

email, ftp und remote login

Ein besonders wichtiges Systemprogramm ist der *Kommandointerpreter* (command interpreter), der die Eingaben des Benutzers entgegennimmt und ihre Ausführung veranlasst. Während die Kommandos früher eingetippt werden mussten, kann heute für viele Aufgaben die Maus eingesetzt werden; so kann man zum Beispiel eine Datei löschen, indem man ihr Bild mit der Maus in einen Papierkorb bewegt. Noch komfortabler ist der Start des zugehörigen Anwendungsprogramms durch Doppelklicken auf eine Datei: so wird bei einer Textdatei das Textprogramm gestartet, das zum Typ der Datei passt, und bei einer Tondatei ein Abspielprogramm; ausführbare Dateien in Maschinencode werden geladen und ausgeführt.

Kommando-interpreter

Früher waren die Programme zur Ausführung der Benutzerkommandos selbst Teil des Kommandointerpreters, wodurch dieser unhandlich und Änderungen mühsam wurden. Heute kann man in UNIX und Linux jedem Benutzer leicht einen maßgeschneiderten Kommandointerpreter – eine sogenannte *shell* – zur Verfügung stellen. Um ein neues Kommando namens *neukommando* einzuführen genügt es, eine Datei mit dem Namen *neukommando* anzulegen, in der das zugehörige ausführbare Systemprogramm steht, und dem System beim Aufruf mitzuteilen, in welchem Verzeichnis sich diese Datei befindet.

1.9 Komplexere Systeme

In den vorangegangenen Abschnitten haben wir angenommen, dass der Rechner nach dem von-Neumann-Modell aufgebaut ist, also nur über eine einzige CPU verfügt; siehe Abschnitt 1.2. Am Ende dieser Kurseinheit wollen wir noch einen kurzen Blick auf weitere Rechnerarchitekturen werfen.

1.9.1 Parallelrechner

Ein naheliegender Gedanke ist es, die Bearbeitungszeit für die Lösung komplexer Probleme dadurch zu verkürzen, dass man die Arbeit auf mehrere Prozessoren verteilt; dieser Ansatz führt zum Konzept des *Parallelrechners*; oft findet sich auch die Bezeichnung *Multiprozessorsystem* (multiprocessor system).

Hier sollte man sich allerdings vor übergroßem Optimismus hüten: Ein Rechner mit k Prozessoren löst ein Problem nicht unbedingt k mal so schnell wie ein Einprozessorsystem![54] Denn zum einen setzt das Zusammenwirken mehrerer Prozessoren *Kommunikation* voraus; auch sie benötigt Zeit. Zum anderen ist zunächst zu prüfen, ob sich ein gegebenes Problem überhaupt gut

Parallelisierung

[54] Dies wird durch alltägliche Erfahrung bestätigt: Ein Team von 30 Programmierern schafft auch nicht an einem Tag, wozu ein einzelner einen Monat braucht.

parallelisieren lässt. Bei Schachproblemen und dem Brechen von Verschlüsselungscodes geht das zum Beispiel recht gut, aber wie steht es mit dem Problem, n Zahlen der Größe nach zu sortieren?[55]

Einsparung

Neben der möglichen Effizienzsteigerung bieten Parallelrechner zwei weitere Vorteile: Indem man viele CPUs in einem gemeinsamen Gehäuse unterbringt, lassen sich Kosten einsparen. Und man gewinnt eine gewisse Sicherheit gegen Störungen; wenn nämlich eine CPU ausfällt, können die anderen ihre Arbeit mit übernehmen. Man kann sogar einen spontan auftretenden Rechenfehler einer CPU neutralisieren, wenn man dieselbe Berechnung zur Kontrolle auch von anderen Prozessoren des Systems durchführen lässt. Solche *fehlertoleranten Systeme* (fault tolerant systems) werden dort eingesetzt, wo es besonders auf Zuverlässigkeit ankommt.

Fehlertoleranz

bus- oder schalterbasiert

Bei einem Parallelrechner teilen sich die Prozessoren einen gemeinsamen großen Hauptspeicher. Der Zugriff erfolgt entweder über einen Bus, oder der gemeinsame Hauptspeicher wird in kleinere Stücke zerlegt und ein *Schalterwerk* sorgt dafür, dass jeder Prozessor auf jeden Teil des Speichers zugreifen kann.

1.9.2 Verteilte Systeme

Bei einem *verteilten System* hat jeder Prozessor seinen eigenen Speicher, auf den nur er selbst zugreifen kann. Die Kommunikation zwischen den Prozessoren kann also nur über den Versand von Nachrichten erfolgen, wie in Abschnitt 1.6 beschrieben. Es kann sein, dass die Prozessoren im selben Rechnergehäuse untergebracht sind und zwischen einigen von ihnen feste Verbindungen bestehen. Ein verteiltes System kann aber auch aus vielen Ein- oder Mehrprozessorrechnern bestehen, die auf der ganzen Welt verteilt sind; dies ist zum Beispiel beim World Wide Web der Fall.[56]

Hypercube

Ein Beispiel für den ersten Fall ist die *Hypercube-Architektur*. Beim d-dimensionalen Hypercube gibt es für jeden der 2^d möglichen Vektoren, die man aus d Nullen und Einsen bilden kann, einen Prozessor. Nun ist aber nicht jeder Prozessor direkt mit jedem anderen verbunden, sondern nur mit den d Prozessoren, deren Vektor sich an genau einer Stelle vom eigenen Vektor unterscheidet. Von (10011000) nach (10011010) gibt es also eine Verbindung, nach (10111010) aber nicht.

Übungsaufgabe 1.20 Über wie viele Zwischenstationen muss man beim d-dimensionalen Hypercube höchstens laufen, um von einem Prozessor zu einem anderen zu gelangen?

`https://e.feu.de/1801-pfadlaenge`

Für verteilte Systeme Software zu entwickeln, ist nicht einfach. Ein Grund liegt darin, dass viele Prozesse gleichzeitig ablaufen, die sich durch Nachrich-

[55] Wer sich für solche Fragen interessiert, sei auf den Kurs *Parallele Algorithmen* verwiesen.

[56] Für die Geschwindigkeit der Kommunikation bedeutet das natürlich einen erheblichen Unterschied.

ten gegenseitig beeinflussen können, dass sich aber nicht genau planen lässt, welcher Prozessor zu welcher Zeit welche Anweisung ausführt.

Wer hierüber mehr erfahren möchte, sei auf die Thematik *Rechnernetze und Verteilte Systeme* und die Kurse *Betriebssysteme*, *Verteilte Systeme* und *Sicherheit im Internet* verwiesen.

1.9.3 Realzeitsysteme

Bei unseren bisherigen Betrachtungen war unser Ziel, ein Programm korrekt ablaufen zu lassen und dabei die Rechenzeit möglichst gering zu halten. Es gibt aber eine ganze Reihe von Anwendungen, bei denen das allein noch keine befriedigende Lösung garantiert. Wenn zum Beispiel die Sensoren eines mobilen Industrieroboters den Kontakt mit einem festen Hindernis melden, muss die eingebaute Steuerung einen Haltebefehl auslösen, bevor der Roboter die Wand durchbricht.

Hier ist es Teil der Korrektheitsanforderung, dass das Ergebnis *innerhalb einer vorgegebenen Zeit* berechnet wird. Mit gängigen Multitaskingsystemen lassen sich solche harten Anforderungen nicht erfüllen; man denke nur an die Folgen, wenn im kritischen Moment die Prozedur zum Halten des Antriebs erst von der Magnetplatte geladen werden muss, oder wenn andere Prozesse mit höherer Priorität die CPU für sich beanspruchen.

Aus diesem Grund gibt es spezielle *Realzeitsysteme*, bei denen große Teile der Software in ROMs untergebracht sind. Die Konstruktion von Realzeitsystemen ist ein interessantes Spezialgebiet.

Literatur

[1] SUSE Supportdatenbank, Updates, Links, Bestellungen etc. http://www.suse.de.

[2] Webressourcen zum Thema Linux, u. a. Liste von Linux-Distributoren. http://www.linux.org.

[3] M. Beck, H. Böhme, M. Dziadzka, U. Kunitz, R. Magnus, C. Schröter, D. Verworner. *Linux Kernel Programming*. Addison-Wesley, 2002.

[4] J. Gulbins, K. Obermayr. *UNIX System V.4: Begriffe, Konzepte, Kommandos, Schnittstellen*. Springer-Verlag, 2012.

[5] M. Kofler. *Linux: Das umfassende Handbuch*. Rheinwerk Computing, 15., überarbeitete Auflage, 2017.

[6] J. Nehmer, P. Sturm. *Systemsoftware – Grundlagen moderner Betriebssysteme*. dpunkt Verlag, 2001.

[7] A. Silberschatz, P. B. Galvin, G. Gagne. *Operating System Concepts*. John Wiley & Sons, 9th edition, 2013.

[8] A. S. Tanenbaum. *Modern Operating Systems*. Pearson Education, 4th edition, 2014.

[9] A. S. Tanenbaum, A. S. Woodhull. *Operating Systems: Design and Implementation*. Prentice Hall, third edition, 2006.

Weitere Weblinks zum Thema Betriebssysteme und Linux finden sie unter

http://www.fernuni-hagen.de/ks/1801/

002683725
(10/21)

01801-6-01-S 1